Detection Strategies: Metrics-Based Rules for Detecting Design Flaws

Radu Marinescu
Department of Computer Science
”Politehnica” University of Timisoara
Bvd. V. Parvan 2, 300223 Timisoara (Romania)
radum@cs.utt.ro

Context of the problems

- There is no perfect software design;
- The **flaws** of the design structure have a **strong** negative **impact** on quality attributes such as **flexibility** and **maintainability**.
- Metrics definitions: imprecise, confusing or incomplete;
  - interpretation of measurement results.
- Metrics are used in isolation: the interpretation of individual measurements is too fine-grained;
  - this reduces the applicability and relevance of metrics usage.
- Interpretation models can offer an understanding of symptoms;
  - But they cannot bring the understanding of the disease that caused the symptoms.

Purpose

- Goal-oriented investigation mechanism, namely detection strategy;
- To define a usable top-down investigation approach;
- Setting rules;
- Model evaluation and validation.

Detection Strategy

- Is the quantifiable expression of a rule by which design fragments that are conforming to that rule can be detected in the source code;
- The use of metrics in the detection strategies is based on the mechanisms of **filtering** and **composition**.

The Filtering Mechanism

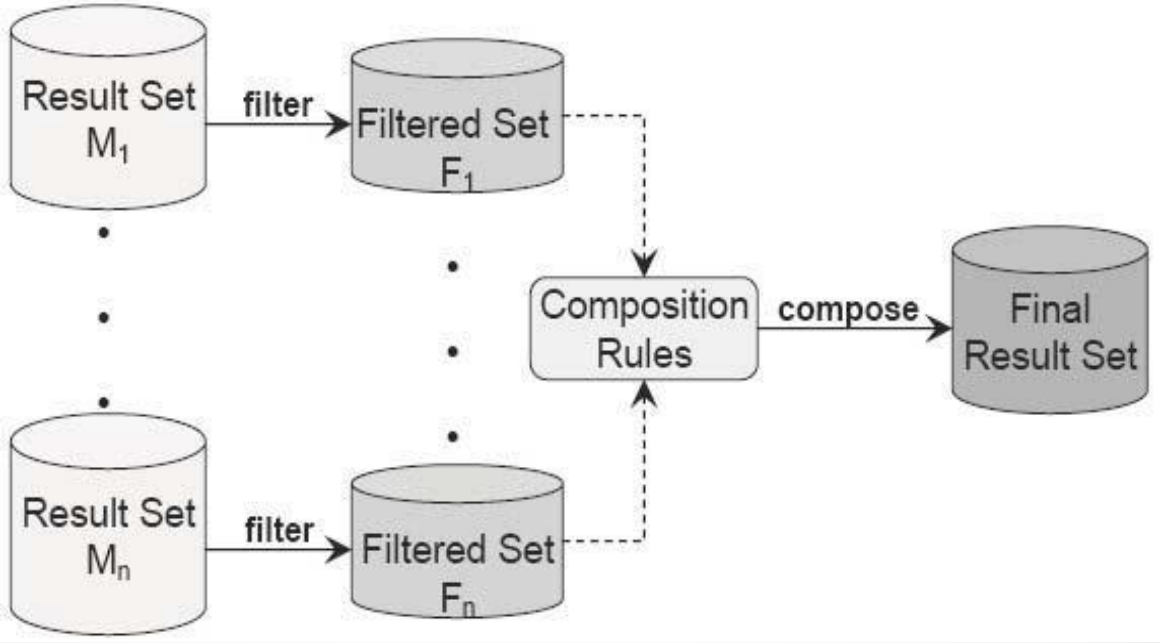
Type of Data Filter	Limit Specifiers		Filter Example
Marginal	Semantical (Limit Value)	Relative	• TopValues (10), • BottomValues (5%)
		Absolute	• HigherThan (20), • LowerThan (6)
	Statistical		• Avoid packages with an excessively high number of classes
Type of Data Filter	**Specfication**		**Filter Example**
Interval	Composition of two marginal filters		Between (20, 30), HigherThan (20) ^ LowerThan (30)

Selecting Data Filters

- Rule 1: Choose an **absolute semantical** filter
  - to quantify the design rules that specify limits explicitly (HigherThan (20)).
- Rule 2: Choose a **relative semantical** filter
  - when the design rule is defined in terms of marginal values, like ”the highest/lowest values”.
- Rule 3:
  - For large systems: Parameterize **relative semantical** filters using percentile values (BottomValues (5%)).
  - For small-scale systems, use **absolute parameters** (HigherThan (20)).
- Rule 4: Choose a **statistical** filter
  - the design rules make reference to extremely high/low values, without specifying any precise threshold.

The Composition Mechanism

- Filtering mechanism that supports the interpretation of individual metric results;
- Second mechanism to support a correlated interpretation of multiple result sets.



Threshold Values

- Aspect that has a **decisive influence** on the **accuracy** of a detection strategy;
- The problem is far from being new and it characterizes an metrics-based approach;
- Means of improvement on this issues:
  - *Experience and Hints from Literature*
  - *Tuning Machine*
  - *Analysis of Multiple Versions*

Threshold Values

- **Experience and Hints from Literature**
  - it is **guided** by similar **past experiences** and by hints from metrics' authors.
- **Tuning Machine**
  - is based on **building** a repository of "**flaw samples**";
  - design fragments that have been **identified by engineers** as being affected by a particular design problem;
  - the examples **repository must be large** enough to allow a tuning;
  - collecting these samples **is not easy.**
- **Analysis of Multiple Versions**
  - information about **the change stability** of a class or the **persistency** of a design flaw over time;
  - is **not to help parameterizing** a strategy with thresholds, but it improves the **accuracy of the detection process**.

Defining a Detection Strategy

- Example: God Classes;
- Heuristics (Riel's):
  - Top-level classes in a design should share work uniformly [...]
  - Beware of classes with non-communicative [...]
  - Beware of classes that access directly data from other classes.
- Stepwise Methodology.

Stepwise Methodology

- 1° Step: Break-down the informal rules in a correlated set of symptoms:
  - Rule 1: refers to a uniform **distribution** of **intelligence** among **classes**, and thus it refers to **high class complexity**;
  - Rule 2: the level of **intra-class communication** between; thus it refers to the **low cohesion** of classes;
  - Rule 3: special type of **coupling**, the direct access to instance variables defined in other classes. In this case the symptom is access of ”**foreign” data**.

Stepwise Methodology

- 2° Step: *select metrics* that quantify best each of the identified properties.
  - *Weighted Method Count* (**WMC**) is the sum of the statical **complexity** of all methods in a class;
  - *Tight Class* ***Cohesion*** (**TCC**) is the relative number of directly connected methods;
  - ***Access to Foreign Data*** **(ATFD)** represents the number of external classes from which a given class accesses attributes, directly or via accessor-methods.

Stepwise Methodology

- 3° Step: find the adequate **filtering mechanism** that should be used with each metric.
  - first symptom is a ”high class complexity”
    - choose the TopValues relative semantical filter for the WMC metric.
  - For the “low cohesion” symptom
    - choose again a relative semantical – BottomValues 1.
  - For the third symptom an absolute filter
    - to capture any access to a foreign data: HigherThan filter.

Stepwise Methodology

- 4° Step: to set the parameters:
  - Simplistic approach;
  - Consider a 25% value:
    - TopValues filter to the WMC metric;
    - BottomValues filter of the TCC metric;
  - For the ATFD metric: no direct access to the data of other classes so the threshold value is 1.

Stepwise Methodology

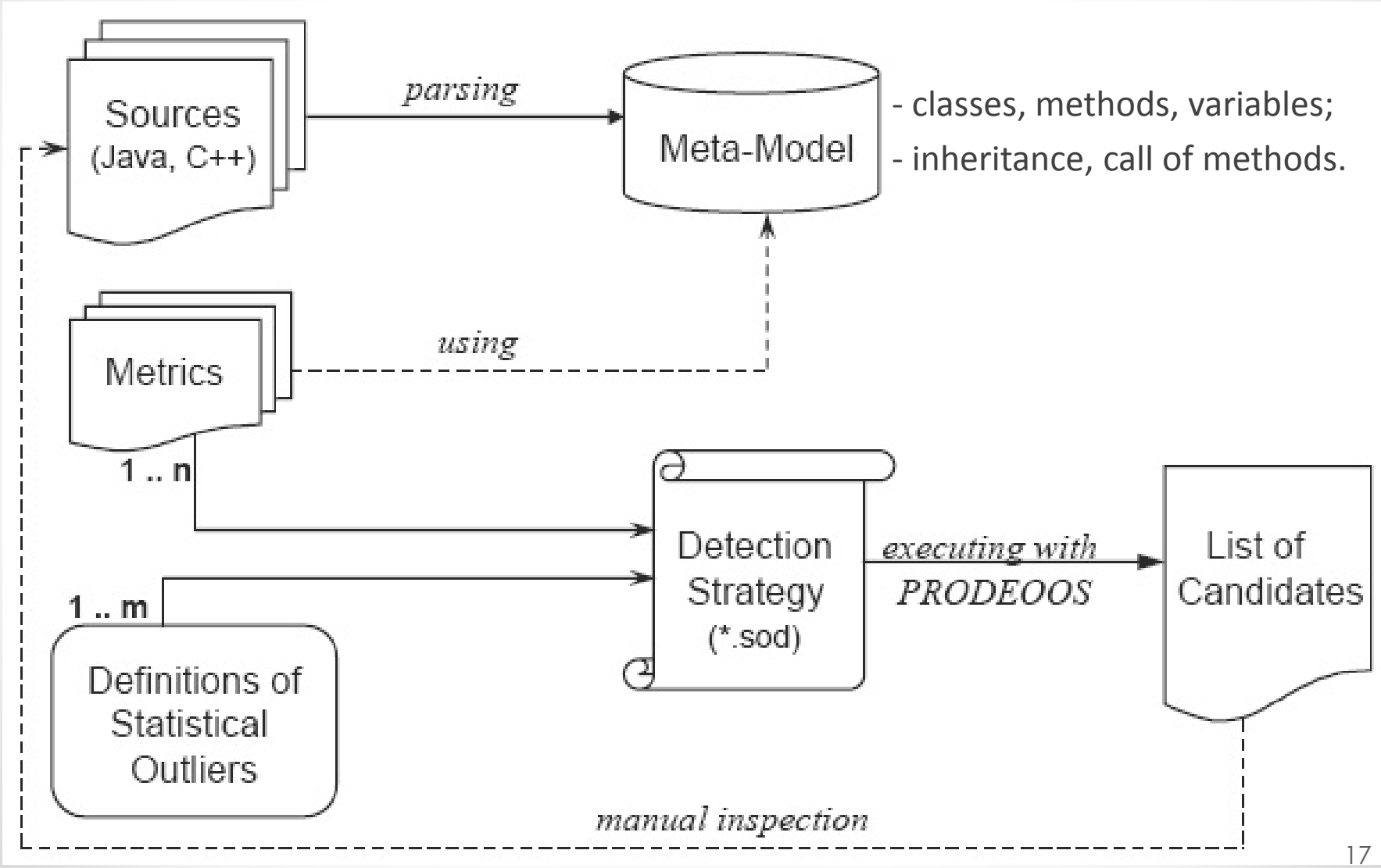
- 5° Step: is to correlate the symptoms, using the composition operators
  - The three symptoms should coexist if a class is to be considered a behavioral God Class;
  - Used the and (^) operator to connect all symptoms.
  - Equation:

$$GodClass(S) = S' \left| \begin{array}{l} S' \subseteq S, \forall C \in S' \\ (WMC(C), TopValues(25\%)) \wedge (ATFD(C), HigherThan(1)) \wedge (TCC, BottomValues(25\%)) \end{array} \right.$$

Implementation of Detection Strategies

- To automatize the execution of detection strategies: interpreted language called SOD.
- The language works with the filtering and composition of metrics results;
  - SQL queries for the computation of individual metrics.

Overview of the inspection process



Evaluation

- **Evaluation of Applicability**
  - ability to quantify design problems at different levels of abstraction;
  - ability to address problems related to key criteria of good design.
- **Evaluation of Accuracy**
  - *Automatic Classification*
  - *Manual Investigation*

System	KLOC	Packages	Classes	Methods
SV1	93	18	152	1284
SV2	115.6	29	387	3446

Results

Design Flaw	Suspects in SV1	Suspects in SV2	False Positives	Special Cases	Real Flaws	Accuracy Rate
Feature Envy	40	15	11	4	25	63%
God Method	4	4	1	3	3	75%
Shotgun Surgery	15	7	6	1	9	60%
Refused Bequest	22	6	4	2	18	81%
God Class	5	2	2	0	3	60%
Data Class	3	2	1	1	2	66%
God Package	2	1	1	0	1	50%
Misplaced Class	4	2	1	1	3	75%
Wide Subsys. Interface	5	1	1	0	4	80%

Results Manual Investigation

Design Flaw	Total Suspects	Correct Detection	Other Flaw	False Positive	Strict Accuracy	Loose Accuracy
God Method	4	2	1	1	50%	75%
Shotgun Surgery	15	10	2	3	66%	80%
God Class	5	3	1	0	80%	100%
Data Class	3	3	0	0	100%	100%
Wide Subsys. Interface	5	3	1	1	60%	80%

Conclusion of the results

- For all the strategies the accuracy rate is over 50%;
- The average accuracy is over 67%;
- The strategy with the lowest accuracy rates (50%) is: *God Package*, which they believe is due to the low absolute number of suspects;
- The conclusion is that the strategies that they defined have a high accuracy in detecting ill-designed entities;
- The process of defining detection strategies is easy to follow.

Future Work

- Address in more detail the issue of choosing threshold values;
- To continue the validation with other software systems;
- There are also design problems that cannot use detection strategies, so they intend to research the possibility of extending the mechanism.

Detection Strategies: Metrics-Based Rules for Detecting Design Flaws

Radu Marinescu
Department of Computer Science
”Politehnica” University of Timisoara
Bvd. V. Parvan 2, 300223 Timisoara (Romania)
radum@cs.utt.ro