Automation Of The Design Flaw Detection Process In Object-Oriented Systems

Ciprian-Bogdan Chirilă
chirila@cs.utt.ro

September 10, 2002

Abstract

Because of the huge number of monolithic and inflexible object-oriented systems and their huge costs there is a need for redesign in order to maintain and to reuse them.

For redesigning such a system we have to eliminate the design flaws, and this is possible with the help of detection strategies in a systematic, scalable and repeatable way.

In this paper will be presented a software tool which modelates the detection strategies using metrics, statistical operators, detects problems in legacy systems and offers introspection at the exact suspect entity.

1 Introduction

In the '80s were built a huge number of object-oriented systems which have to be developed further. Using the object-oriented technique advantages are achieved better software quality and reduced development time.

In the '90s were obtained a huge number of object-oriented systems on large scale which seem to be:

- *inflexible*: there can not be added very easily new functionalities
- *monolithic*: there is a lack of structured functionality for the system which is based on components
- *hard to maintain*: adding new features fails in an endless chain of changes in multiple places.

A decision has to be made about such a software system. If dealing with a low economic value system and the system is not flexible we may drop it. The problem is critical in the situation of high economical value systems. The development have to continue in spite of it's inflexibility. There is a need for *flaws detection* in order to eliminate them and to continue the development process.

2 Flaw Detection Automation

We built a *software tool* which will offer a *general* and *scalable* detection process. The approach is based on the detection strategy

concept defined in the articles [6, Mari01a], [7, Mari01b] which reffer to design flaw detection based on metrics.

2.1 Metamodel

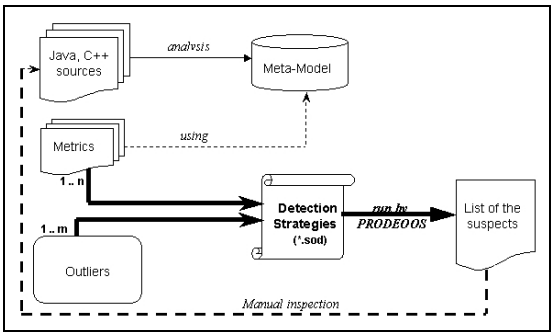

Figure 1: Approach

Starting from the source code of a legacy system is generated a metamodel which contains information that assists the detection process. The informations included are accesses, method calls, class declarations, variable declarations, inheritance relationships informations which are stored in tables.

2.2 Metrics

2.2.1 Definition

Metrics are functions that reflect the properties of the software entities through a number. By entities are reffered classes, methods and subsystems.

2.2.2 Example

Weight of a Class (WOC) By *Weight of a Class* metric is measured the ratio of the non set-get methods and all the members in the interface of the class. The metric counts the functionality degree of the class.

Number of Public Attributes (NOPA) The name of the metric is very suggestive. The result is useful for the detection of classes which stores data separating them from their behaviour, an error-prone in object-oriented technology.

Number of Accessor Methods (NOAM) The accessor methods we are reffering to are the set-get methods which have the job of reading or writing values into class members. Some programmers hide data in classes separating them for their natural behaviour.

2.3 Outliers

2.3.1 Definition

An outlier is a statistical operator which applied to a data collection, extracts the extreme value data. The software tool will apply outliers on entity collections. The outlier is in fact a filtering mechanism that will select the suspect entities.

The parametrization of the defined mechanisms is a complex problem. The current approach is based on setting a default initial value for the parameters. During the case studies those parameters will vary but finally they will stop at a given value or range. The magnitude of the analyzed software project must be considered when tuning outliers.

2.3.2 Example

Top Values, Bottom Values This type of outliers are interested in extreme value entities. Entities with extreme values measured may be considered suspicious.

Higher Than Values, Lower Than Values This type of outliers filter those entities with values higher or lower than a selected threshold. Entities with that type of values may be considered suspicious.

BoxPlots The BoxPlots outlier is a complex filtering mechanism used for compairing two nearly-continuous variables.

2.4 Detection Strategy

2.4.1 Definition

The definition for the detection strategy of a design flaw is: [6] cuantifiable expresion of a rule which may detect automatically the entities affected by the design flaw. The detection strategy is a mechanism based on concepts like **filtering** and **composition**. The rule may be formalized by a metric based formula:

$$S := {M_1}^{O_1} * {M_2}^{O_2} * \cdots * {M_n}^{O_n}$$

$$* := \cup \quad | \quad \cap \quad | \quad \setminus$$

In order to operate over the metamodel a tree structure to model the theoretical formula is needed. This set of (simplified) rules is presented here:

```
DetectionStrategy
 := StrategyDefinition
    SymbolsDefinition

# rules for defining a
  detection strategy
StrategyDefinition
 := StrategyName ":="
    DetectionRule ";"

DetectionRule
 := MetricWithOutliers |
    ComposedDetectionRule

MetricWithOutliers
 := "(" MetricName ","
     OutlierName ")"

ComposedDetectionRule
 := DetectionRule
    CompositionOperator
    DetectionRule

StrategyName
 := [A-z][A-z0-9_]

CompositionOperator
 := "or" |
    "and" |
    "butnotin"

# symbols are defined by
  metrics and outliers
SymbolsDefinition
 := MetricDefinition |
    OutlierDefinition

# rules for metric definitions
MetricDefinition
 := MetricName
    ":="
    SqlQuery ";"

MetricName
 := [A-z][A-z0-9_]
```

```
SqlQuery
 := [.]
    # SELECT <Entity> <Value>

# rules for outlier
  definitions
OutlierDefinition
 := OutlierName
    ":=" OutlierType
    "(" OutlierParameter ")" ";"

OutlierType
 := "TopValues" |
    "BottomValues" |
    "HigherThan"|
    "LowerThan" |
    "BoxPlots"

OutlierName
 := [A-z][A-z0-9_]

OutlierParameter
 := [0-9][0-9,][%]
```

2.4.2 Example

Data Classes The example proposed is a detection strategy which detects data-classes. Data-classes are ??? [7, Mari01b].

```
DataClassesStrategy:=
(
 (WOC, WOCBottom) and
 (WOC, WOCLower) and
 ((NOPA, NOPAOutliers) or
  (NOAM, NOAMOutliers))
);

WOC:=
      SELECT all_pub.f_class
      AS f_class,
      (100*pub_mth.cnt /
       all_pub.cnt)
      AS woc FROM...
NOPA:=
      SELECT f_class,
      count(f_class)
      AS nopa
      FROM v_members...
NOAM:=
      SELECT f_class,
      count(F_class)
      AS noam
      FROM v_accessor_methods...

WOCBottom    := BottomValues(10);
WOCLower     := LowerThan(33);
NOPAOutliers := TopValues(7);
NOAMOutliers := TopValues(5);
```

We will tackle about this strategy and it's target to detect data classes. By data classes we mean those classes which separates data from their behaviour. Based on this principle we may build the strategy in the folowing way: we are interested in those classes that have many data members and do not have much functionality. The strategy will be composed out of WOC metric in order to measure the lack of functionality, NOPA and NOAM metrics. ???

Case Study We run "Data Classes" strategy on a case study and the folowing result showed up:

Class	WOC	NOPA	NOAM
A	0.05	14	
B	0.06	16	
C	0.10		20

The results are the following: class A and B are lighter than the threshold imposed by the outlier and we may say that are data classes. Class C doesn't have public members instead it is hiding them under accessor methods.

3 Conclusions

The benefits using the software tool are:

- *sistematic approach* by the description of the strategies using the SOD language. With the SOD language one may create several strategies that will incapsulate the experience and the effort used in a manual design flaw detection.

- *repeatability* based on the fact that the strategies are reusable, the paramaters may change but the strategy remains the same. The tunning of the parameters should be made in conformance with the scale of the software system. By using the parameters also the degree of acuracy or severity of the flaw detection may be set.

- *scalability* provided by the software tool PRODEOOS and the results obtained on industrial systems. The software tool behaves the same way on a large scale system as on a small scale system. The only thing that differs in those cases is the detection time. After a detection has made the user has direct access to the suspect entities of the system, which will accelerate the process of understanding the cause of the flaw.

References

[1] H. Schildt: *C++, Manual complet*, Editura TEORA, 1997.

[2] G. Booch: *Object-Oriented Analysis and Design with Applications*, Second Edition, Addison-Wesley, 1994.

[3] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1997.

[4] Robert Martin: *Design Principles and Patterns.* http://www.objectmentor.com, 2000.

[5] Fowler Martin: *Refactoring.* Second Edition, Addison-Wesley, 1999.

[6] R. Marinescu. *Detecting Design Flaws via Metrics in Object-Oriented Systems*. Proceedings of the TOOLS-USA 39, ISBN 0-7695-1251-8, IEEE Computer Society, 2001.

[7] R. Marinescu. *Design Flaws and Detection Strategies*. PhD Presentation, Karlsruhe, April 2001.