The meta-architecture of the detection strategies tuning machine

Mihancea Petru Florin

LOOSE Research Group, “Politehnica” University of Timisoara, Romania

pepy@cs.utt.ro

Abstract

As a proverb says "Use numbers, but never thrust them". The current method for building detection strategies in order to detect OOP design flaws raises a big question: because the quality of a detection strategy depends strongly on the proper parameterization of the filtering mechanism, how can we establish the right parameters? Unfortunately, we don't have a clear formal method that we can use it in the parameterization process and we are therefore using informal methods based on the experience of the designer. To avoid this situation, we aim to build a machine that can analyze a large number of problem examples and based on that, help us establish the best parameters. In this position paper we will describe the meta-architecture of this machine. As a result, we will have a proper overview regarding the components of such a machine, its basic functionality, and the other necessary elements.

1. The problem

Defining a new detection strategy consists of the following two steps [1]: the analysis of the informal rule and the selection of metrics. In the second step we need to select those metrics that are most relevant to the strategy obtained in the first step. At the same time, for each selected metric we must provide its statistical function that represents the filtering mechanism. The metric and its statistical function compose the metric expression. Combining these expressions by using composition operators, we will obtain the detection strategy skell, a detection strategy expression without any parameters for the filtering mechanism. All the statistical functions must be parameterized in a proper way because the quality of the detection strategy strongly depends on this parameterization. Figure 1 sum up all these activities.

The selection process of the metrics is an easy one. That's because this activity is based on the strategy obtained by the analysis of the informal rule. The strategy describes very clear, in a domain specific language, the properties of the problem that we want to detect. In many cases, these properties directly denote the metrics that we must use and also indicate the statistical functions associated with the selected metrics. Unfortunately, selecting the parameters for the filtering mechanism is not so simple. "Mapping the problem into numbers" is a real problem when we create new detection strategies. Let's consider an example for the detection of an OOP design flaw. We want to detect DataClasses. As R. Marinescu wrote in [2] the strategy looks like this:

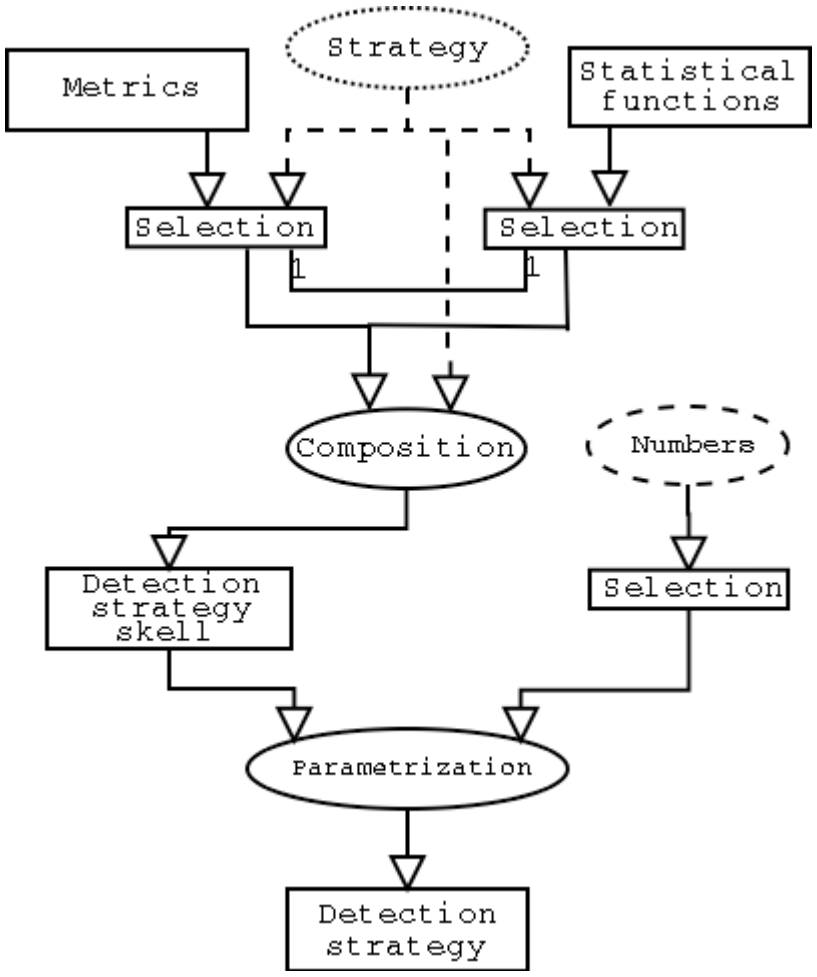

Figure 1- Defining a detection strategy: step 2.

"We will detect data-classes based on their characteristics: we search for "lightweight" classes, i.e. classes which provides almost no functionality through their interface. Next, we will look for the classes that define many accessor methods (get/set methods) and for those who declare data fields in their interface. Finally, we will confront the lists and manually inspect the "lightweight" classes that declare many public attributes and those who do provide many accessor methods."

Passing this strategy through the process described in the figure 1, the detection strategy rule looks like this:

DataClasses:=((WOC,BottomValues(33%))and(WOC,LowerThan(0.33)))and((NOPA,HigherThan(5)) or (NOAM, HigherThan(5)))

We can observe that for a "lightweight" class the value for the WOC software metric (weight of a class) must be lower that 0.33 and also, that if we sort in an ascending order

the WOC of all the analyzed classes, it must be in the first 33% values of all values. At this point some questions arise: Can we consider that a class with a WOC value of 0.40 or 0.34 is not a "lightweight" class? Why we can consider that a "lightweight" class with a NOPA (number of public attributes) value of 4 and a NOAM (number of accessor methods) value of 4 is not a DataClass?

The problem that generates these questions is that we use constant values as parameters of the statistical functions, values that are not obtained as a result of formal selection reasoning. As an example, today, when we are defining a new detection strategy for an OOP design flaw, the parameterization process is based entirely on our experience regarding the software design. That means that we analyze a set of examples regarding the problem that we want to detect and as a result, we establish the parameters in such a manner that the detection strategy can detect all the examples as problem. This parameterization method raises some issues:

- We don't have a precise description of how this analysis must be done
- The analysis is made manually, so the set of examples can not be a large one
- Because the set of examples is small, it is created by a small number of persons that could know each other, so the examples could have a big subjective component
- It is very difficult to include all the particularities of a problem in a small set of examples
- An example of a problem may be correct in a context, but incorrect in others, so in a new context we must change some examples from the set and analyze the new one. This operation takes time because the analysis is made manually.

In this position paper we will describe at a pure abstract level a new parameterization method that will help us to avoid all the problems described above.

2. Attacking the problem

Our idea is to build a machine, a tool that will be able to automatically analyze a large number of examples of a specific problem and help us establish the best parameters for the filtering mechanism of the detection strategy used to detect that problem. To reach this goal, first of all we must describe, at a pure abstract level, the general aspects of this machine. This is absolutely necessary because detection strategies may be applied in distinct domains, so our detection strategies tuning machine may have some particularities dependent of some domain knowledge.

In this position paper, our goal is to define all the notions involved in a description of a detection strategies tuning machine. We must describe all the essential components of a machine, components that any implementation of the detection strategy tuning machine must have. Some of the aspects of a component may of course differ from a real machine to another, depending on the implementation context, but in the end we have just one component that we talk about. At the same time, we must describe the connections between these components and the basic functionality of a detection strategies tuning

machine. All these enumerated elements constitute the meta-architecture of the detection strategies tuning machine.

3. The Meta - Architecture

3.1. "Repository" component

Definition. The "Repository" is a component of the detection strategies tuning machine meta-architecture responsible for all the operations used to manage the examples.

This component consists of two main subcomponents: Data repository and Descriptors repository. We must define some new notions before explaining what these subcomponents do.

Definition. A data sample is an indivisible entity, completely or incompletely specified, that can compose a problem completely, alone or together with other data samples.

Definition. A descriptor is an entity that completely describes an example.

Using these two new definitions we can define the subcomponents of the "Repository":

Definition. The data repository is a subcomponent of the "Repository" responsible for all the operations used to manage the data samples.

Definition. The descriptors repository is a subcomponent of the "Repository" responsible for all the operations used to manage the descriptors.

As we can see from the figure 2, an example contains a descriptor and one or more data samples. An important observation is that the same data sample may be used in distinct examples, reducing the dimension of the data repository. The descriptor will contain information like the name of the problem or the problem ID that uniquely identify the problem found in the described example, identifiers for all the data samples included in the example described, and other additional information. The problem ID will be used to identify the set of examples that will be analyzed by the machine to adjust the parameters of a detection strategy that detects the problem identified by the same problem ID.

Let's consider now a real machine used for the tuning of a detection strategy that detects an OOP design flaw. A data sample will be in this case a class. Data repository will manage files, each file containing a class. The class may not be completely defined, but we must be able to calculate correctly on it those software metrics that are required for the detection strategy tuning. This possibility will reduce the example dimension. Do not forget that we will work with many examples, so their dimension is critical. The descriptor may be a file that contains the description information. A data sample will be identified by the path and name of the file that contains the class code.

3.2. "Descriptors analyzer" component

Definition. The "Descriptors analyzer" is a component of the detection strategies tuning machine meta-architecture responsible for verifying from the lexical, syntactical and semantically point of view, the description information contained by the descriptors of all the examples that compose a set, and for translating these information in a form recognized by the tuner component.

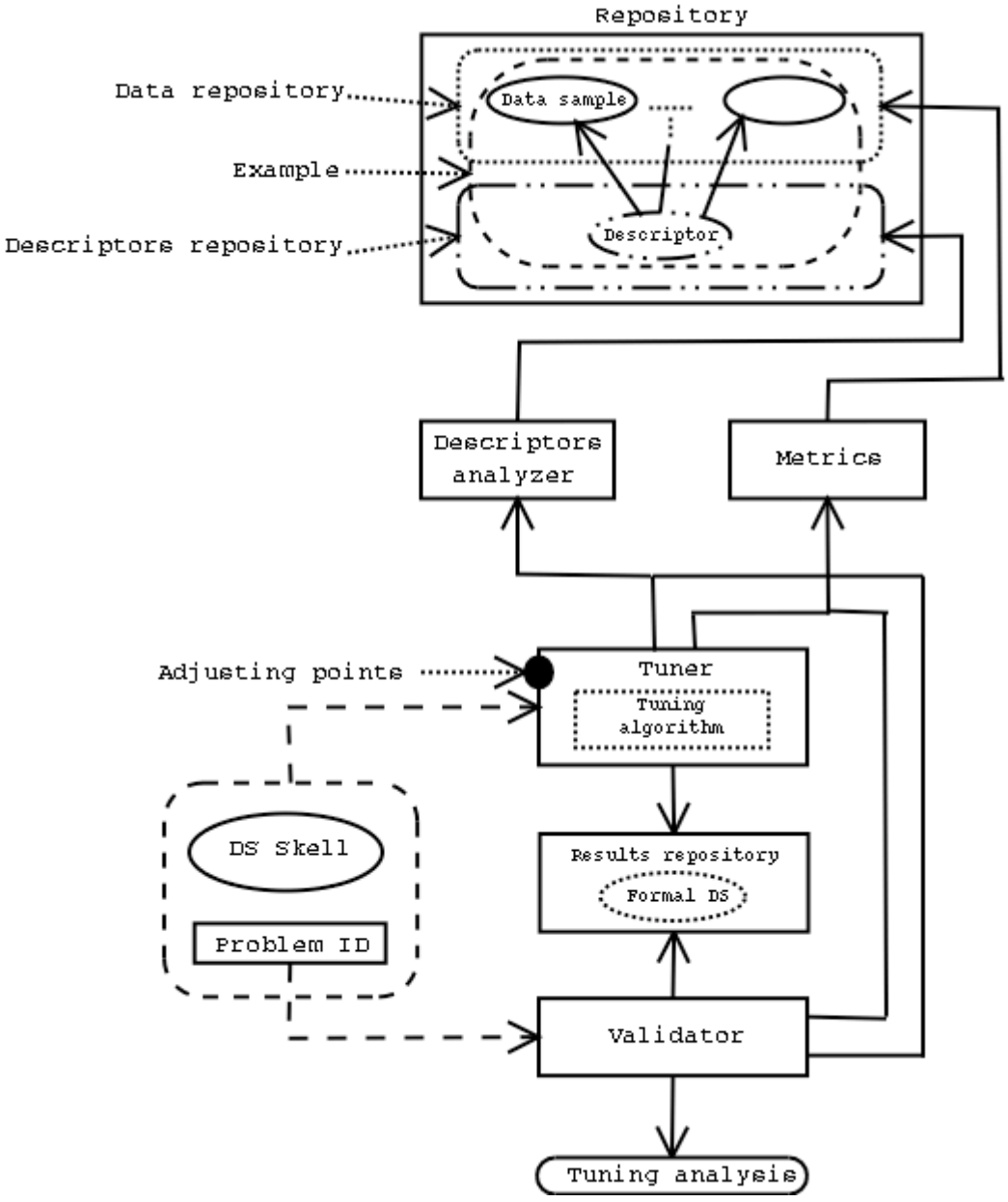

Figure 2 - The meta-architecture of the detection strategies tuning machine

The complexity of this component depends strongly on the descriptor specification language. If we have a simple descriptor that contains only an enumeration of the problem name and of the data samples identifiers, the analyzer may be a very simple one.

The complexity increases if the descriptor is an XML file and it contains a lots of others additional information. The specification language is established when a real tuning machine is implemented, depending on the descriptor content. The additional information that a descriptor must contain depends on the specific knowledge domain where the machine will be functioning. The "Descriptors analyzer" is used by the tuner component to access the set of examples that are necessary for the tuning of the detection strategy.

3.3. "Metrics" component

Definition. The "Metrics" is a component of the detection strategies tuning machine meta-architecture that is able to measure an attribute of the problem on a concrete example of problem. If measurement errors may appear this component must be able to deal such situations.

This component depends strongly on the specific domain knowledge where we use detection strategies as problems detection techniques. Considering the example where we use detection strategies to detect OOP design flaw, the "Metrics" component will be a subsystem of the tuning machine that will be able to calculate some software metrics over the data samples that compose an example of problem.

Let's consider now that we want to use a detection strategy to detect if a person has a certain disease. The "Metrics" component of the tuning machine that will adjust the parameters of the detection strategy will be able to make some medical analysis (metrics) on a person (example) that has this disease. It can do that by searching some records (data samples) of the person in a data base (data repository). For each metric the detection strategies tuning machine will search for a limit value that, if surpassed, shows that the suspected person may have the disease that we are looking for (i.e. the tuning machine says that the temperature of the suspected person must be greater than 39 degree Celsius if the disease is present). As we can see, the software metrics and the medical analysis are similar from the detection strategies tuning machine point of view.

3.4. "Tuner" component

Definition. The "Tuner" is a component of the detection strategies tuning machine meta-architecture that tunes the parameters of a detection strategy base on a tuning algorithm.

As we can see in figure 2, the main subcomponent is the tuning algorithm. It can use statistical, artificial intelligence or other techniques to adjust the parameters of the detection strategy. The quality of the parameters established by the detection strategies tuning machine strongly depends on this tuning algorithm.

Definition. An adjusting point is a user-machine interface through which the user can influence the tuning process.

The tuner component may have zero, one or more adjusting points depending on the tuning algorithm and the interactions possibilities permitted by it. Any real tuning

machine must have at least one adjusting point. The tuning process is based on a big number of examples. The tuning machine will use a set of examples to find the best parameters for the detection strategy being tuned. If we apply the detection strategy that we have obtained before on the same set used by the machine in the tuning process, it is possible to miss to identify as problems all the examples from the set. That's because the constraints imposed by the algorithm are too powerful. We say that we have some false negative examples, real problems that are not detected by the detection strategy. Ideally, the number of false negative examples must be zero. In the real world this is not possible on all occasions, but we can minimize this number.

Definition. Trust level is an adjusting point of the tuner component that can be used by the user in order to reduce or increase the number of false negative examples, reducing or increasing the constraints imposed by the tuning algorithm.

As we can see from the figure 2, the tuner component receives the skell of the detection strategy that will be tuned. Therefore the tuner component may have a subcomponent used to analyze the detection strategy skell. This is not included in the meta-architecture because it is not relevant from the tuning process point of view.

3.5. "Results repository" component

Before defining this component we must introduce a new notion.

Definition. A formal detection strategy is a detection strategy, which was parameterized by a detection strategies tuning machine.

Definition. The "Results repository" is a component of the detection strategies tuning machine meta-architecture responsible for managing a history list of formal detection strategies obtained by the machine.

In its simplest form, this component will contain only the last formal detection strategy tuned by the machine. Maintaining in this repository all the formal detection strategies ever tuned by our machine, will give us the possibility to analyze the performances of a formal detection strategy versus another formal detection strategy constructed on the same skell but on different sets of examples.

3.6. "Validator" component

Definition. The "Validator" is a component of the detection strategies tuning machine meta-architecture responsible for comparing the performances of all formal detection strategies that detect the same problem and that are stored in the results repository and for indicating the best one.

Using this component we can compare the performances of a formal detection strategy tuned using a set with another formal detection strategy constructed on the same skell and tuned using the same set but introducing some new examples in it. In this manner, we can

observe if the examples from the initial set completely include all the particularities of the problem. We can also compare two formal detection strategy constructed on differed skells but tuned on the same set. In this manner we can see if the metric selection process and statistical function selection process can be optimized. Of course, the formal detection strategies that are compared detect the same problem. Many more other tests may be imagined.

3.7. How does it work?

The first thing that must be done before starting the tuning process is the construction of the set of examples that describe the problem. Using the informal rule we will detect if an entity or a group of entities are "attacked by a disease" or, in other words, have the problem described by the rule. When we detect a situation like this we consider that we have an example of the problem we are looking for. The entities are decomposed in data samples, which are stored in the data repository. After that we write the descriptor that contains information about the data samples included in the example and the problem ID. The descriptor is stored in the descriptor repository. We proceed in the same manner for each example that we have found. When we consider that the set comprises a sufficient number of examples and that these examples contain all the particularities of the problem, we can start the tuning process.

The detection strategy skell and the problem ID are user inputs for the tuner component. The tuner to identify which metrics must be calculated and what statistical function must be parameterized analyzes the skell. After that, the tuner component will ask the descriptors analyzer component to find in the descriptors repository descriptors that describe examples with a problem ID equals with the problem ID gave by user. All the selected descriptors will be analyzed, translated in a form recognizable by the tuner component and provided to it. For each example the tuner component will ask the metrics component to calculate all the necessary metrics. The metrics component will access the data repository to resolve a request. The measurement results will be used by the tuning algorithm to adjust the parameters of the statistical functions included in the detection strategy being tuned. If necessary, the tuner component may access the results repository component to have a look at other formal detection strategies tuned by this machine and that detect a problem identified by the same problem ID. When the tuning process is done, the tuner fills the parameters from the detection strategy skell and the new formal detection strategy is stored in the results repository.

Next, the control is transferred to the validator component. The activation is made by the user or by the tuner component but again, this is not relevant from the tuning process point of view. The validator must have as an input a problem ID that identifies all the formal detection strategies that will be tested and that are stored in the results repository. The problem ID will be also used to select the set of examples used for validation. The validator component will use the descriptors analyzer component in the same manner like the tuner component. Each formal detection strategy will be applied on all the examples from the selected set. To calculate the metrics the validator will use the metrics component. We want to apply a formal detection strategy on all the examples from the set

because we want to see how many examples are really detected as problems. All the examples that are not detected as problems are false negative examples because all the examples from the set are instances of the same problem. We need to apply all the formal detection strategies on the same set because we want to find the best one that is, as a possible rule, the formal detection strategy with the smallest number of false negative examples. All the information about the performance of each formal detection strategy and the best one found at the moment are written in the tuning analysis report.

We will unfortunately have situations when the best formal detection strategy reported will present an inadequate number of false negative examples. This situation may appear when the formal detection strategy has an incomplete skell. If we want to use the same skell but we need a formal detection strategy with a smaller number of false negative examples we can repeat the tuning process by reducing the tuning algorithm constraints. To do this, the user will interact with the tuning algorithm using the trust level adjusting point. Reducing the algorithm constraints it is possible to reduce the number of false negative examples, but at the same time the number of false positive suspects will increase. We say that a suspect detected by a detection strategy is a false positive suspect if it does not present the problem detected by the detection strategy used. Of course, this is not as inconvenient as having a big number of false negative example because it is not grave to suspect an entity having a problem that it does not have, but it is very grave to not detect an entity having a problem when it really presents that problem. This is of course a temporary solution for reducing the number of false negative examples. The best way to reduce this number is to create a new detection strategy skell.

4. Conclusions

In this position paper we described a new method for the parameterization of detection strategies. The detection strategy tuning machine is a tool that will receive as inputs a detection strategy skell and a problem ID that identify the problem detected by the detection strategy being tuned. The machine will analyze all the examples of problem with the same problem ID, will calculate some metrics on these examples and will adjust the parameters of the detection strategy. As a result, the machine will return the best detection strategy for that moment.

The detection strategies can be applied in different domains. Because of that, their form will be influenced by the specific domain knowledge. The detection strategies tuning machine will be also influenced in the same manner. In this position paper, we describe the detection strategy tuning machine at pure abstract level. The meta-architecture shows us the essential components of detection strategies tuning machine, their responsibilities and the basic functionality of the entire machine. Of course, any implementation of this meta-architecture may include new elements if the developer considers that this is necessary but all real detection strategies tuning machine can be reduced to the meta-architecture described in this position paper.

A machine created on this meta-architecture will eliminate all the major problems that the current method of parameterization has. First of all, the parameterization is made in an

automated way. Because of that, we can use a bigger number of examples. Next, we have a clear description of the reasoning applied for parameters adjustments. The tuning algorithm is responsible for that. Finally, because we can process a large number of examples, the number of persons that will select these examples may be a very large one (i.e. a computer programming community). We can provide them with the informal rule of the problem for selecting the examples and independently creating the detection strategy skell. Because the persons who selects the examples do not know the form of the detection strategy skell, we can avoid a big subjective influence in the examples selection process.

References

1. Radu Marinescu, “Measurement and Quality in Object-Oriented Design”, PhD Report #3, Timisoara, 2002.
2. Radu Marinescu, “Detection Strategies”, September 24, 2001.