Prioritization of Artifacts for Unit Testing Using Genetic Algorithm Multi-objective Non Pareto

Eduardo Noronha de Andrade Freitas¹, Auri Marcelo Rizzo Vincenzi², and Celso Gonçalves Camilo-Júnior²
¹Department of Informatic, Instituto Federal de Goiás, Goiânia, Goiás, Brazil
²Institute of Informatic, Universidade Federal de Goiás, Goiânia, Goiás, Brazil

Abstract—The choice of units for application of write unit testing in level of methods can be seen as a combinatorial problem. The time demanded to write unit tests becomes a challenge for the testing professional considering the constraints of time existent. Adopting the time to write unit testing as a cost, the goal this work is to present a multi-objective evolutionary approach that look for a subset of artifacts such that minimizes the cost at the same time that maximizes their strategic importance. Both metrics, static and dynamic such as cyclomatic complexity, operational coverage, and frequency of modification were used to define the strategic importance. The motivation to use multi-objective evolutionary approach is that growth of fault proneness is inversely proportional to the cost. The experiments were done in real context of industry system, and the results found confirmed the benefits of proposal.

Keywords: Software testing, unit testing, effort test, prioritization unit test, Search Based Software Test (SBST), multiobjective evolutionary optimization.

1. Introduction

Among the activities of the Software Engineering, the verification and validation are the more expensive, representing 50% to 80% of the total cost of a project, and software testing is the most common practice for software verification and validation.

The development of unit testing in this context is an especial challenge for the professionals who find themselves in difficulty in deciding which artifacts must be the firsts, once the resources available to conclude this activity are limited. In this sense, the development and the application of unit test on the whole system with high level of coverage may be impractical, if considered the resources constraint. The identification of artifacts really relevant of system become essential and strategic. It is especially in the case of legacy systems, large systems, and systems with high level of maintenance.

In this work, the problem of prioritization of effort of unit tests will be discussed with techniques of Search Based Software Engineering (SBSE) [7]. In SBSE the problems are modeled as optimization problem, and after that they are solve utilizing concepts, techniques, algorithms, and strategies of search. The objective is to identify among all possible solutions a set of solutions, which will be sufficiently good according to a set of appropriate metrics.

The validation of the proposed approach was carried out some software companies seeking a process of real experimentation in context of high criticality. Some decision variables were considered to the experimentation of the algorithms such as: cyclomatic complexity, frequency of changes found on repository, and operational coverage. The constraint of the problem will be considered in terms of hours, considering the availability of time to the development of unit tests to homologation of a new release of the system.

The rest of the paper is organized as follows: In the Section 2 is presented the concept of software testing and evolutionary optimization utilized to the development this work. In the Section 3 is described the characterization of the problem of selecting code units and its complexity. In the Section 4 is presented the details of the approach and of the experiments realized. Finally, in the Section 5 is presented the conclusion of the work developed.

2. Fundaments

The goal this Section is to present part of the concepts utilized in the development of the work.

2.1 Software Testing

According Delamaro et al. [5], the software testing consists in an activity of quality assurance in order to verify if the product in development is in conformity with your specification. The [8] defines software testing as a dynamic verification of the program’s behavior, utilizing a set of finite test, correctly selected of the executions’ domain, to verify if it’s according with the expected.

Many techniques and test’s criteria had been developed to permit the selection of a subset of the entrance domain to be effective in reveal the presence of the existent defects. This occurs because mostly of time the exhaustive test is impracticable due to constraints of time and cost.

According to Pressman [9], the testing activity can be considered as a incremental activity realized in three phases: unit testing, integration testing, and high level testing.

Unit tests, also called of unit testing has its focus in the minor code unit in order to ensure that the implementation’s
aspects are correct. In this phase, seeks to ensure greater coverage and maximum defect’s detection in each module.

The scope this work is focused on building an alternative to find among the artifacts at the unit level, which should be selected to the development of structural tests in the method level. The prioritization must be carried out considering the possibility of combining two or more code’s unit as priority for a specific moment. So, this work doesn’t focus the prioritization of test cases.

2.2 Evolutionary Optimization

In mathematically or in computation, the term optimization is used as reference to study of problems that seek maximize or minimize a function by choosing values to the variables of the problem.

However, some problems own combinatorial characteristic and can be computationally intractable due the your dimension of exponential order. In that case, the classical optimization models become limited to present an optimal solution in feasible time.

The concept of evolutionary computation has been used by means of so-called evolutionary algorithms. They apply the process of natural evolution defended by Charles Darwin, as paradigm to implementation of the algorithms. The evolutionary heuristics give up the warranty of global optimal for ensuring a set of approximated solutions.

Some problems are considered as multiobjective. A multi-objective problem consists in finding a set of variables that satisfy some constraints, while optimizing two or more objectives. The solution of a multi-objective problem is composed by a set of solutions that represent a commitment among the objectives, according to Azuma [1].

According to Coello [2], a multi-objective optimization problem can be formulated as:

\[
\min\{f_1(x), f_2(x), \ldots, f_k(x)\}
\]

subject to \(m\) inequality constraints:

\[
g_i(x) \leq 0 \quad i = 1, 2, \ldots, m
\]

and to \(p\) equality constraints:

\[
h_i(x) = 0 \quad i = 1, 2, \ldots, m
\]

where \(k\) is the number of functions, \(e x = [x_1, x_1, \ldots, x_n]^T\) the array of decision’s variables. The problem becomes to be to determine values among the set \(F\) of all the array’s variables, to satisfy the Equations 2 e 3, the particular set of values \(x_1, x_2, \ldots, x_n\) that provide optimal values of all functions. So, in context of prioritization of code unit, a solution is considered as a set of code units.

One important concept in multi-objective optimization is associated the dominance of the solutions. A solution \(x\) dominates another solution \(y\), if both condition follows happen in a optimization problem: if the solution \(x\) is better or equals to \(y\) in all objective functions, and The solution \(x\) is \(\epsilon\) strictly better than \(y\) in at least one objective function.

There are two different spaces in multi-objective optimization. The first one is associated with the variable of the problem, while the second is associated with the objectives.

Each point enumerated in space of the variables correspond to code units of a arbitrary system. The mapping these points in solutions in objective space can be comprehended by a strong analogy to classic Knapsack Problem. The decision consists in choice the better combination of methods to be exercised in unit testing. To do it, the function of mapping will go utilizing some metrics that they contain strong correlation interdependent with strategic importance, according to will be detailed in Section 4.

In Figure 1 are presented some of these solutions. Each solution present in the objective space may have 3 distinct classifications: the first when in which the solution dominates, is dominated, and when it is irrelevant. In Figure 1 a small example of mapping the space of objectives, containing 2 goals is presented. On the first objective we seek to find a set of code units with high values of strategic importance associated. In the second objective the aim is to find a set of code units that require the least amount of time to develop unit tests. These requirements confer to the problem a multi-objective characterization, once mostly cases the higher the strategic importance of a unit takes longer time for the development of unit testing.

Fig. 1: Analysis of the space of objectives, considering time and cyclomatic complexity.

The solution A shown in Figure 1 represents a set of code units that after the mapping to the space solution presents the value 3 to complexity, requiring 1 unit of time (for example, hours) to the development of unit testing.

Some methods are used to measure the objective functions such as the weighted sum of the objectives, and also, the method \(\epsilon\)-restrict. In this work was adopted a multi-objective evolutionary approach with a genetic algorithm that uses weighted sum.s
3. The Problem

The objective this Section is to present the characterization of the problem of prioritization of software artifacts in level of methods or function, and the challenge inherent to it.

In the context of software testing, specifically unit testing, we seek to find a subset of code units (methods) that together maximize the strategic importance in software. Another objective consists in minimizing the time used to developing test cases to these code units. The existence these two conflicting objectives justifies the necessity of the use of SBST to solve this problem, especially with multi-objective approach.

To calculate all distinct ways to choice elements without repetition is necessary to realize the sum of distinct combinations of \( r \) elements, according to Equation 4.

\[
\sum_{r=1}^{n} \binom{n}{r} = \frac{n!}{r!(n-r)!}
\]  

(4)

To illustrate the exponential behavior of the problem, the Figure 2 shows the curve containing the number of possible combinations to systems with different quantity of code units.

Fig. 2: Exponential growth possibilities for selection of units for unit testing code.

In Section 4 will be presented the main characteristics of a multi-objective optimization problem, and how to prioritize units of code to the development of unit testing can be solved by this approach.

4. Proposed Approach

The activity of software testing has revealed challenging problems. Identifying strategic interesting areas to reduce the effort in testing is one of them. In Elberzhager et al. [6] is presented a systematic mapping of existing approaches able of reducing the effort in testing, one being the prediction of areas with most prone to defects. The essence that approach is the consideration of metrics. The key metrics adopted in that studies reviewed were: product, process, object oriented, and defects.

Also, others approaches to minimize the effort in testing were found. In Shihab [11] is presented a comparative analysis of some heuristics in order to prioritize software artifacts for unit testing in legacy systems. According to the study, the application of unit tests on artifacts that have undergone corrective changes more frequently, have greater potential to reveal defects. Other approach is found in Ray and Mohapatra [10] in which a method is proposed to prioritize testing effort in order to guide the tester during the development life cycle of software. Amount of influence, average execution time, structural complexity, and severity and value, were considered relevant factors. In the work of Czerwonka et al. [4] a system for failure prediction, risk analysis and prioritization of tests is presented to supporting the aspects of maintaining legacy systems from Microsoft, particularly Windows Vista. However, it is also applied prioritization of test cases.

The main shortcoming with these approaches is that it isn’t seen as a combinatorial problem. The works that focus on identifying priority artifacts to application of unit testing, using simple heuristics, whose result is a list of artifacts in descending order of importance. A trend in predicting defects is explores the metrics and their combinations. However, a small number of papers have focused the combination of static and dynamics metrics.

The objective this section is to characterize the proposal of prioritization of code based in multi-objetive evolutionary approach. The research made use of experiments to validate the proposal. This section shows the context and strategy for prioritization followed by a detailed of the experiments utilizing Genetic Algorithm Multiobjective Non Pareto using aggregating functions of same weights [3], combining all objectives into a single one.

The proposed approach to selecting units of code for application of unit testing is supported in Search Based Software Engineering (SBSE) more specifically Search Based Software Testing (SBST). Considering a system with several modules, its dependencies, and a great quantity of methods, the test’s planning for a new release must be optimized, considering limited resources for the development of the testing activies.

The amount of metrics available to be used to establishing a strategy to optimize the selection of units of code is very large. For this work was defined as the first objective function the maximization of strategic importance of the units selected. The main goal is to try to find out a selection of units (methods) with higher strategic importance. Basically, three main components are used to address it: cyclomatic complexity, operational coverage, and frequency of modification.

The mensuration of the strategic importance (\( si \)) of each
method is calculated based on the product and process metrics. Some works influenced the choice of the metrics utilized by algorithm, as Ray and Mohapatra [10] which have showed some important metrics such as time of execution, influence’s value of a component, structural complexity, responsibility per class and LCOM4, type and severity of failures. In Shihab et al. [11], the authors present a comparison of some heuristics to prioritize artifacts to application of unit testing in legacy systems. The application of unit testing in artifacts that receive corrective changes most frequently own more potential to reveal failures. Thus, we aim at to figure out which characteristics exist among units more recently corrected. This analyze amongst the evidences shown in [6] concluded that they own high values of cyclomatic complexity. Based in this, the cyclomatic complexity was used initially as a general guide for defect-prone.

The following metrics were considered to calculating the strategic importance of a subset of units of code for application of testing activities, mainly unit testing: Cyclomatic Complexity, Number of corrective changes, and Operational Coverage that measures the intensity the artifacts are exercised in operational environment.

Some techniques and technologies were used to extract metrics to permit the execution of techniques of prioritization. To obtain the quantity of changes was utilized the software svnstat and the software jdiff. Also, was developed a software to realize a parser in the files of the repository to shows how many times an artifact was changed, and what the intensity that modification. The intensity was measured by number of lines of code changed.

The operational coverage was found utilizing the software Cobertura, and execution of an instrumented version in production environment. This metric was considered highly important, once it allows identifying which lines of code are really executed by user, and the frequency of utilization. Thus, even an artifact which metrics that indicates the needed of writing unit testing, it is discouraged if no actual use in a production environment occurs. On the other hand, an artifact can be strongly recommended to be tested, if it is very exercised in a production environment.

All metrics were extracted and persisted in a database modeled specifically to allow the heuristics in their executions. The Figure 3 illustrates this process.

The first objective function is defined according to Equation 5.

\[ s_i = \sum_{i=1}^{N} (NScc(i) \times Wcc(i)) + (NSoc(i) \times Woc(i)) + (NSfm(i) \times Wfm(i)) \] (5)

where:
- \( s_i \): Strategic importance;
- \( N \): Quantity of units (methods) in the system;
- \( NScc \): Normalized score of cyclomatic complexity of the method;
- \( NSoc \): Normalized value of operational coverage of the method;
- \( NSfm \): Normalized value of Frequency of Modification on repository of version of the method;
- \( Wcc \): Weight considered for cyclomatic complexity;
- \( Woc \): Weight considered for operational coverage;
- \( Wfm \): Weight considered for Frequency of Modification on repository of version;
- \( x \): It receives value 1 if the artifact \( i \) has been considered, and 0 otherwise.

For each artifact the normalized scores \( NS (NScc, NSoc, NSfm) \) are calculated as in order to assign values between 0 and 100, according to example shown in Table 2. There are 2 kind of metric: bigger is better, and smaller is better. For the first one, the normalization is given by the following equation:

\[ NS = \frac{(mv - min)}{(max - min)} \times 100 \] (6)

As an example, the metric of cyclomatic complexity presented in the Table 2, the higher its value is more interesting to choose this artifact to the application of unit testing. For the metric type the smaller is better, the normalization is performed by the following equation:

\[ NS = 1 - \frac{(mv - min)}{(max - min)} \times 100 \] (7)

where:
- \( NS \): Normalized score;
- \( mv \): Absolute value of metric;
- \( min \): Lowest Absolute value of the metric between units;
- \( max \): Largest absolute value of the metric between units;

Each component \((NScc, NSoc, NSfm)\) assumes a weight, and the sum of the weights must be equal to 1.

Therefore, the selected units will be those whose combination maximizes the objective function presented. Logically, if there is no restriction, all artifacts will be selected for...
employment of unit testing. However, this work owns as second objective function the equation below:

\[
ar = \frac{\sum_{i=1}^{N} t(i) \times x(i) - ha}{ha}
\]

where:

- \( ar \): Availability of resources
- \( t(i) \): time required for the development of unit testing activities (planning, implementation and execution) for the method \( i \);
- \( ha \): hours available.

The calculation of time \( t(i) \) is done based on the number of lines of code, and assuming an productivity average, 1 hour for every 10 lines of code. Therefore, if a method has 180 lines of code, the estimated time for the development of unit activity for this test method will be 18 hours.

The objective is not to exhaust the discussion on what are the best strategic components for selecting units of code, but provide flexibility to the software testing community to choose on different environment metrics that they consider appropriate in the selection model.

In order to address some penalty for the selection that violates the time constraint, the fitness function may be calculated as:

\[
F = si + (si \times ar)
\]

5. Experiments

For these experiments was considered critical software of industry responsible for the integration of a chain of services in accounting and tax segment. The software analyzed was developed in Java and has 808 methods, totaling 11,300 lines of code.

To obtain the operational coverage of the software was generated an instrumented version of the system. This version was put in a production environment for 4 months, and approximately 4,000 users used it during this period of time. Information for the last three months of the versions of each artifact to calculate the frequency of changes in repository were collected. The cyclomatic complexity of each artifact was obtained from the latest stable version of the system, using the software Sonar and metrics plugin.

Table 1: Example of Representation of a Individual.

| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

Table 2: Model of Normalization of Metrics.

<table>
<thead>
<tr>
<th>Artifact</th>
<th>Complexity</th>
<th>Normalized Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>57.1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>14.3</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>28.6</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>85.7</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
<td>100.0</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>14.3</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>57.1</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>28.6</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>100.0</td>
</tr>
</tbody>
</table>

unit test is 300 hours, and the best combination of artifacts found by a heuristic demand 285 hours, 15 hours can be considered in this case as a waste of resources.

The strategy chosen for representation of the algorithm is binary. Each individual has \( n \) genes, and each gene can assume the values 0 or 1, indicating the presence or absence of an artifact in the solution candidate. The value of \( n \) is the number of methods in the project being evaluated.

Initially, the algorithm randomly generates the initial population by assigning each gene, a value: 0 or 1, according to the adopted representation. After that it holds the evolution of the population for generations. In each generation individuals are initially evaluated and assigned fitness to each one.

According to the representation of individuals presented in Table 1, this representation considers the presence of 6 artifact (positions 1, 4, 5, 6, 9 and 10) concerning genes with value 1. Using the metric complexity of these artifacts as an example, the model of normalization presents the values shown in Table 2.

Once evaluated individuals, the feasibility verification of each one is accomplished by ensuring that the time available for the development of testing activities is sufficient to test the artifacts represented in each individual. In cases which an individual violates the constraint of time available, one or more artifacts are removed from the individual, with the objective of making it feasible.

The algorithm worked with operators of crossover of 1 point, mutation operator per gene, and method of the roulette as selection operator to crossover, and the application of elitism value of 1 for the canonical algorithm.

5.1 Analysis of Results

The Figure 4 shows the average behavior of the standard deviation of the average time required for the development of unit testing to the artifacts prioritized. The behavior of the curve can be explained by the EA attempt to optimize the combination of artifacts. With a smaller amount of time available to allocate heuristic search in prioritizing artifacts
with low amount of lines of code because they allow fine-
tune the complementation.

In the first executions, whose available time is reduced, the selection of artifacts that consume more time for de-
volution is practically impossible, and allocating smaller artifacts to use the time available. This behavior is repeated until the heuristic can optimize your choice.

![Fig. 4: Standard Deviation of Size of Artifact.](image)

![Fig. 5: Analises of Loss of Time.](image)

The Figure 5 shows the performance of the heuristics compared to the non-use of the resource of time available, here called waste. The comparison chart shows that the first 50 simulations considered by heuristic called here by SVN [11], the EA proposed has been more effective in 31 of them (62 %). In all the last 50 simulations EA presented, the EA proposed own better results, totaling 81 simulations with better gain in relation to another strategy.

Another important feature in the first simulation 50 is related to the ability to complement the list of prioritized using artifacts from those, which have a small number of lines of code. This wide existence of artifacts with this feature allows the so-called waste to be less.

Figure 6 shows the cyclomatic complexity of the solutions during the experiments. It is noticed that the presence of artifacts with greater complexity is perceived only from the experiment with over 20% of available time. The artifacts with this characteristic are avoided in the presence of low availability of resources, since there is a direct correlation between cyclomatic complexity and number of lines of code.

![Fig. 6: Characteristics of complexity of the methods during the Experiments.](image)

6. Conclusions

The main goal this work was present a new approach to prioritize artifacts in level of methods to application of unit testing. The approach is based in Multi-objective Evolutionary Algorithms as work of SBST. The framework proposed is guided by a flexible and dynamic set of metrics according to interest of the professional. In this work were considered 3 metrics: cyclomatic complexity, operational coverage, and frequency of changes on version’s repository. The initial results were compared with one research in literature. We hope to experiment others techniques of multiobjective evolutionary algorithm as SPEA and NGSA.

We are conducting a research to try to figure out which metrics owns the better power to be used by this framework to reveal defects. Techniques of data mining have been applied to address it. The initial results has been promising. Also, the experimentation of the implementation of EA in parallel environment is being tested.

7. Acknowledge

Grateful thanks are owed to the company Oobj Technology of the Information by its invaluable help in sharing the data, systems, and some workers for the development this work. This work was partially supported by FAPEG.

References


