W7

Track
4/28/2010 1:45 PM

"Automated Test Case Generation Using Classification Trees"

Presented by:

Peter M. Kruse & Magdalena Luniak
Berner & Mattner Systemtechnik GmbH

Brought to you by:

SOFTWARE QUALITY ENGINEERING

330 Corporate Way, Suite 300, Orange Park, FL 32073
888-268-8770 · 904-278-0524 · sqeinfo@sqe.com · www.sqe.com

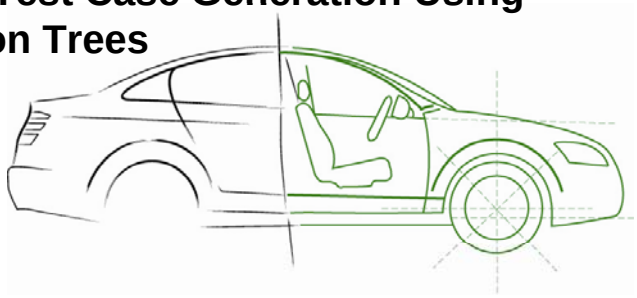
Peter M. Kruse

Peter M. Kruse is a software engineer working in the domain of testing, including evolutionary testing and the classification tree method. He is an experienced software developer and tester in the German automotive industry. Peter's project experience includes Hardware-in-the-Loop Testing (HiL), model driven-development (MDD), and evolutionary structural and functional testing. He is responsible for development of CTE XL, a very popular test design tool.

Magdalena Luniak

Magdalena Luniak's project experience includes development of statistical analysis tools. She has transferred her experience from data analysis into the field of software testing, working on the enhancement of the classification tree method at Berner & Mattner. Magdalena has been involved in software development at StataCorp LP in Texas, as well as teaching computer science at the Technical University Berlin, Germany.

Automated Test Case Generation Using Classification Trees



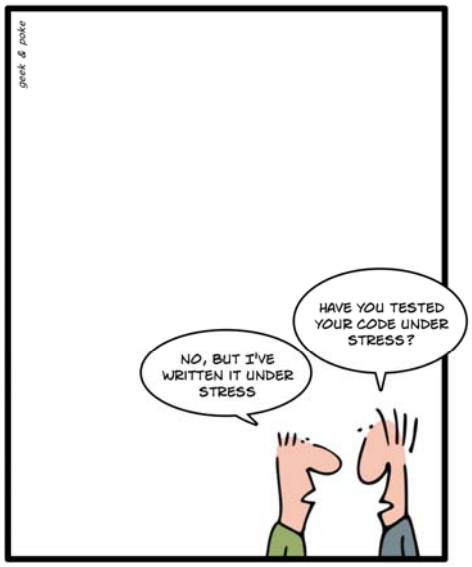
Peter M. Kruse
April 28th, 2010

Contents

berner & mattner
optimizing your development

Introduction

- Problem: Test phase
  - often starts too late,
  - gets truncated,
  - and is therefore too short.
- Question:
  - *How can a "good" subset of test cases be selected?*
  - *How can promising test cases be identified?*
  - *How can the number of test cases be reduced?*



http://geekandpoke.typepad.com/geekandpoke/2009/03/stress-test.html

berner & mattner
optimizing your development

Contents

4

berner & mattner
optimizing your development

Classification tree method

- Method for black-box test case design

[Grimm, Grochtmann 1993]

- **Step 1**: Identify test-relevant aspects
  - functional specification of the test object
  - for each aspect
    - divide the input domain into disjoint subsets
  - resulting in the classification tree

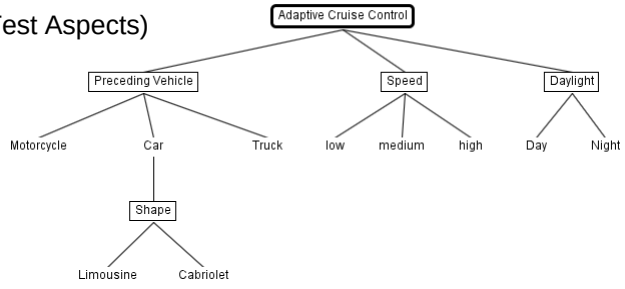
- **Step 2**: Build test cases
  - combine classes from different classifications

berner & mattner
optimizing your development

Classification tree method, step 1

- **System under test**
  Adaptive Cruise Control

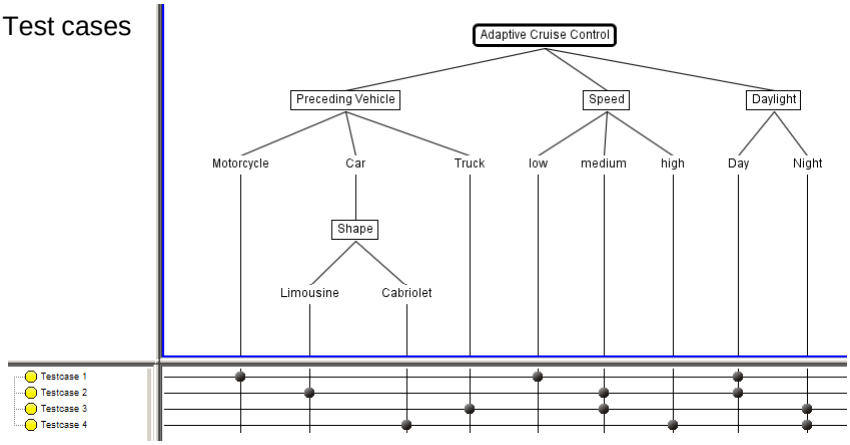
- **Classifications** (Test Aspects)
  Preceding Vehicle
  Speed
  Daylight



- **Classes** (Disjoint Input Values)
  Motorcycle, Car, Limousine, Cabriolet, Truck, low, medium, high, Day, Night

Classification tree method, step 2

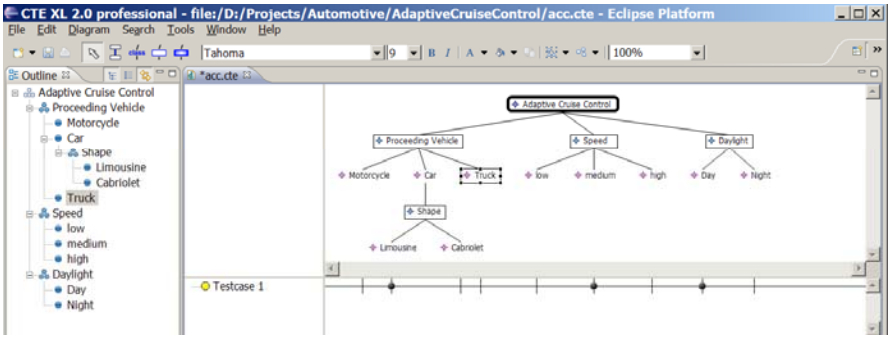
- Test cases



7

Classification tree editor CTE XL

CTE XL 2.0



8

Classification tree editor CTE XL

- Automated test case generation
  - Minimal combination
  - Pairwise combination
  - Threewise combination (“*triplet-wise*”)
  - Complete combination
- User defined dependency rules
- Integration with requirement and test management tools
  (e.g. IBM Rational DOORS®, HP Quality Center®)
- Available for free from http://www.berner-mattner.com/ → *Products*

9

Problem

- Test suite reduction is only possible using different combinations, e.g. pairwise combination instead of complete combination
- No distinction between “good” and “bad” test cases
- No ordering of test cases within a test suite

10

Contents

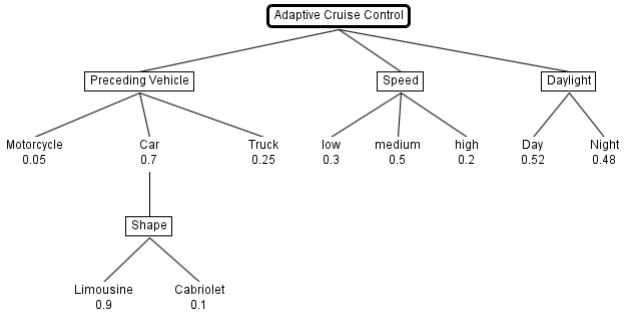
- Introduction
- State of the art
- **Improvements**
  - **Prioritization**
  - **New combination rules**
  - **Coverage criteria**
  - **Optimizing test suites**
- Implementation
- Conclusion

Prioritization

- According to
  - **Usage Model** [Walton, Poore, Trammell 1995]
  - **Error Model** [Elbaum, Malishevsky, Rothermel 2002]
  - **Risk Model** [Amland 2000]
- Assign values to all classes in the tree

Prioritization

- Example: Occurrence probability values in usage model
  - Sum per classification: 1



  - Conditional probability is dissolved using Bayes' theorem

13

New combination rules

- Introducing
  - ***prioritized*** minimal combination (PMC)
  - ***prioritized*** pairwise combination (PPC)
  - class bases ***statistical*** combination (CSC)
- Class weights for occurrence, failure and risk are taken into account
- Dependency rules are also taken into account

14

Prioritized combinations

- Prioritized minimal combination (PMC)
  - Extended minimal combination
  - Each new test case covers the most important remaining class
- Prioritized pairwise combination (PPC)
  - Extended pairwise combination
  - Each new test case covers the most important remaining class pair.
- Test cases are generated in descending order of importance
- Deterministic test suite generation

15

Statistical combinations

- Class bases statistical combination (CSC)
  - Generates test suite with a user defined size
  - Classes are selected by the probability of their weights
- Goal:
  - Test suites which reflect the class weight distribution well
- Random process
- Allows testing of non-deterministic systems
- Test suite may contain redundant test cases

16

Occurrence weights for class pairs

		Day	Night	low	medium	high
		0.52	0.48	0.3	0.5	0.2
Motorcycle	0.05	0.026	0.024	0.015	0.025	0.01
Limousine	0.63	0.3276	0.3024	0.189	0.315	0.126
Cabriolet	0.07	0.0364	0.0336	0.021	0.035	0.014
Truck	0.25	0.13	0.12	0.075	0.125	0.05
low	0.3	0.156	0.144	/	/	/
medium	0.5	0.26	0.24	/	/	/
high	0.2	0.104	0.096	/	/	/

17

Prioritized pairwise combination result

#	Preceding Vehicle	Speed	Daylight
1	Limousine	medium	Day
2	Limousine	medium	Night
3	Limousine	low	Day
4	Truck	low	Night
5	Truck	high	Day
6	Limousine	high	Night
7	Truck	medium	Night
8	Cabriolet	medium	Day
9	Cabriolet	low	Night
10	Motorcycle	medium	Day
11	Motorcycle	low	Night
12	Cabriolet	high	Night
13	Motorcycle	high	Night

18

Coverage criteria

- For prioritized **minimal** combination
  - Each Used Coverage (EUC)

$$EUC = \frac{number\ of\ covered\ classes}{number\ of\ coverable\ classes}$$

  - Weight Coverage (WC)

$$WC = \frac{sum\ of\ weights\ of\ covered\ classes}{sum\ of\ weights\ of\ all\ coverable\ classes}$$

19

Coverage criteria

- For prioritized **pairwise** combination
  - Each Used Coverage (EUC)

$$EUC = \frac{number\ of\ covered\ class\ pairs}{number\ of\ coverable\ class\ pairs}$$

  - Weight Coverage (WC)

$$WC = \frac{sum\ of\ weights\ of\ covered\ class\ pairs}{sum\ of\ weights\ of\ all\ coverable\ class\ pairs}$$

20

Coverage criteria

- For statistical combinations
  - no coverage criterion due to random nature
  - quality assessment based on *chi-square* test available

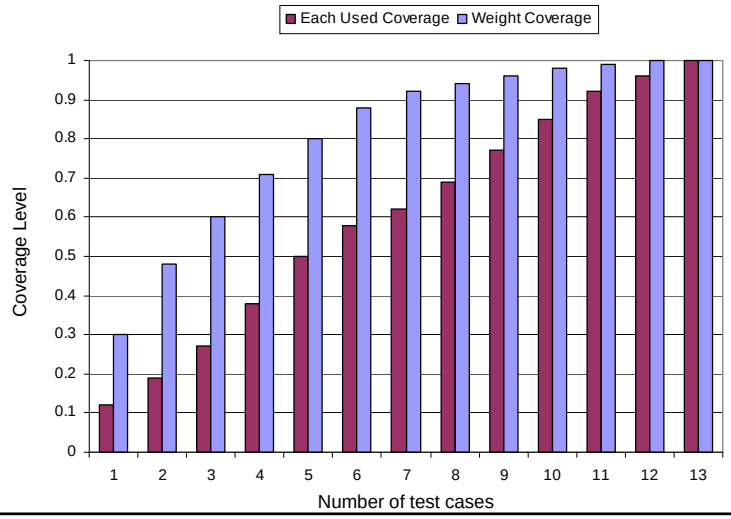
21

Coverage criteria

#	Preceding Vehicle	Speed	Daylight	EUC	WC
1	Limousine	medium	Day	0.12	0.30
2	Limousine	medium	Night	0.19	0.48
3	Limousine	low	Day	0.27	0.60
4	Truck	low	Night	0.38	0.71
5	Truck	high	Day	0.50	0.80
6	Limousine	high	Night	0.58	0.88
7	Truck	medium	Night	0.62	0.92
8	Cabriolet	medium	Day	0.69	0.94
9	Cabriolet	low	Night	0.77	0.96
10	Motorcycle	medium	Day	0.85	0.98
11	Motorcycle	low	Night	0.92	0.99
12	Cabriolet	high	Night	0.96	0.99
13	Motorcycle	high	Night	1.00	1.00

22

Coverage criteria



23

Optimizing test suites

- Selection of test cases based on the
  - coverage criteria
  - coverage level
- Selection of a subset of test cases based on the tester's requirements
- Reduction in the total number of test cases

24

Optimizing test suites

#	Preceding Vehicle	Speed	Daylight	EUC	WC
1	Limousine	medium	Day	0.12	0.30
2	Limousine	medium	Night	0.19	0.48
3	Limousine	low	Day	0.27	0.60
4	Truck	low	Night	0.33	0.71
5	Truck	high	Day	0.50	0.80
6	Limousine	high	Night	0.53	0.88
7	Truck	medium	Night	0.62	0.92
8	Cabriolet	medium	Day	0.69	0.94
9	Cabriolet	low	Night	0.77	0.96
10	Motorcycle	medium	Day	0.85	0.98
11	Motorcycle	low	Night	0.92	0.99
12	Cabriolet	high	Night	0.96	0.99
13	Motorcycle	high	Night	1.00	1.00

30% weight coverage with one test case

Optimizing test suites

#	Preceding Vehicle	Speed	Daylight	EUC	WC
1	Limousine	medium	Day	0.12	0.30
2	Limousine	medium	Night	0.19	0.48
3	Limousine	low	Day	0.27	0.60
4	Truck	low	Night	0.33	0.71
5	Truck	high	Day	0.50	0.80
6	Limousine	high	Night	0.53	0.88
7	Truck	medium	Night	0.62	0.92
8	Cabriolet	medium	Day	0.69	0.94
9	Cabriolet	low	Night	0.77	0.96
10	Motorcycle	medium	Day	0.85	0.98
11	Motorcycle	low	Night	0.92	0.99
12	Cabriolet	high	Night	0.96	0.99
13	Motorcycle	high	Night	1.00	1.00

60% weight coverage with only three test cases

Optimizing test suites

#	Preceding Vehicle	Speed	Daylight	EUC	WC
1	Limousine	medium	Day	0.12	0.30
2	Limousine	medium	Night	0.19	0.48
3	Limousine	low	Day	0.27	0.60
4	Truck	low	Night	0.38	0.71
5	Truck	high	Day	0.50	0.80
6	Limousine	high	Night	0.58	0.88
7	Truck	medium	Night	0.62	0.92
8	Cabriolet	medium	Day	0.69	0.94
9	Cabriolet	low	Night	0.77	0.96
10	Motorcycle	medium	Day	0.85	0.98
11	Motorcycle	low	Night	0.92	0.99
12	Cabriolet	high	Night	0.96	0.99
13	Motorcycle	high	Night	1.00	1.00

50% class pair coverage with only five test cases

27

Optimizing test suites

#	Preceding Vehicle	Speed	Daylight	EUC	WC
1	Limousine	medium	Day	0.12	0.30
2	Limousine	medium	Night	0.19	0.48
3	Limousine	low	Day	0.27	0.60
4	Truck	low	Night	0.38	0.71
5	Truck	high	Day	0.50	0.80
6	Limousine	high	Night	0.58	0.88
7	Truck	medium	Night	0.62	0.92
8	Cabriolet	medium	Day	0.69	0.94
9	Cabriolet	low	Night	0.77	0.96
10	Motorcycle	medium	Day	0.85	0.98
11	Motorcycle	low	Night	0.92	0.99
12	Cabriolet	high	Night	0.96	0.99
13	Motorcycle	high	Night	1.00	1.00

90% weight coverage with just seven test cases

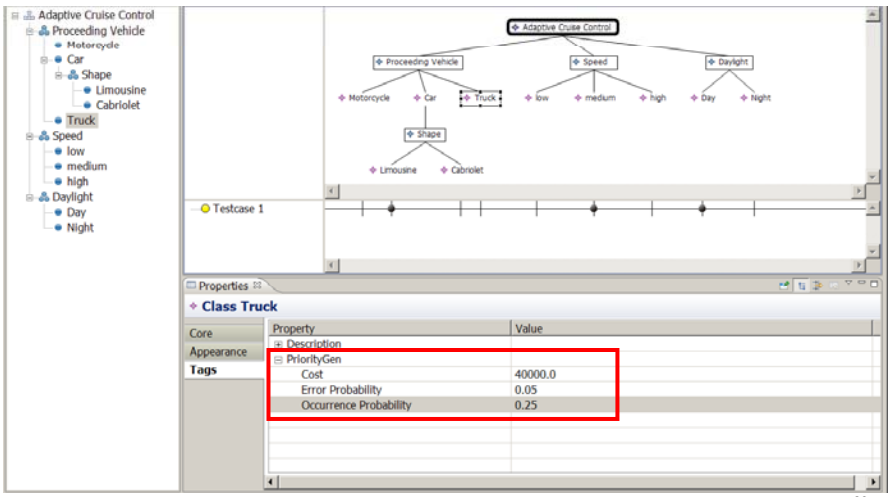
28

Contents

- Introduction
- State of the art
- Improvements
- **Implementation**
  - **Prioritization**
  - **Automated test case generation**
  - **Coverage criteria**
  - **Optimizing test suites**
- Conclusion

29

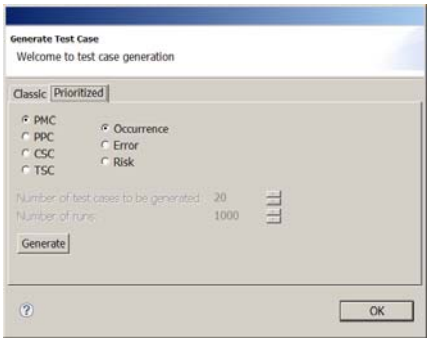
Prioritization



30

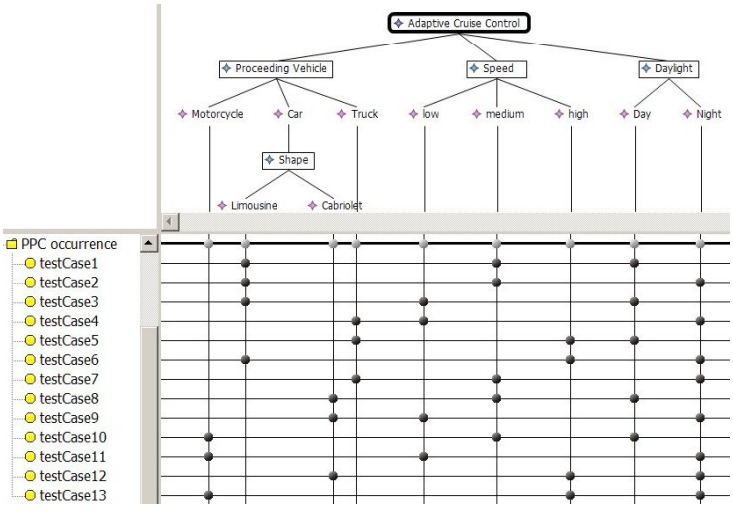
Automated test case generation

- Simple interface
- Easy selection of preferred combination rule
- For statistical combinations:
  - Selection of test set sizes



31

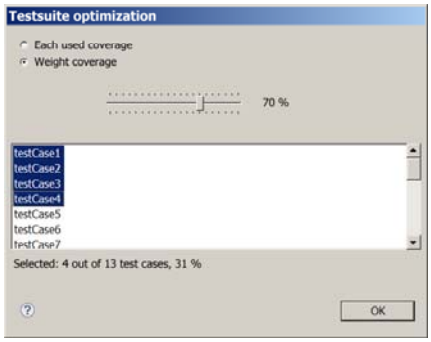
Automated test case generation



32

Optimizing test suites

- Reducing number of test cases
- Based on selected criterion
- Based on preferred coverage
- Interactive preview



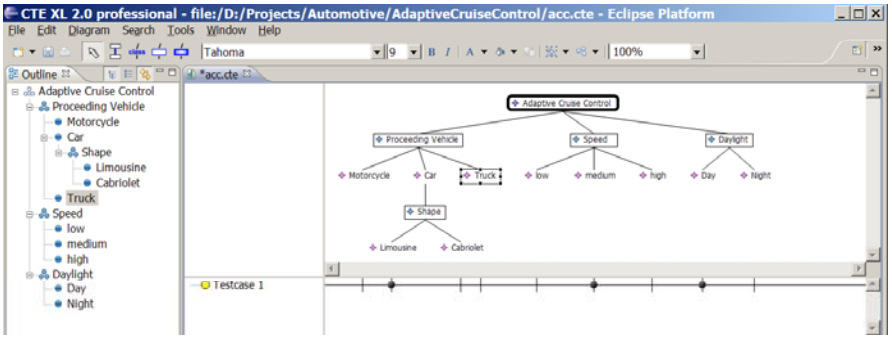
33

Conclusion

- We introduced
  - weighted classes in classification trees
  - new combination rules (both prioritized and statistical)
  - coverage criteria
- This allows
  - identification of profitable test cases
  - selection of good subsets of test cases
  - reduction of test suite size
- Profit gained
  - Generation of optimized, prioritized test suites

34

Conclusion



CTE XL 2.0 Professional

- Available summer 2010

35

References

[Grimm, Grochtmann 1993] Classification Trees for Partition Testing. *Software Testing, Verification & Reliability*, 3(2):63–82, 1993.

[Walton, Poore, Trammell 1995] Statistical testing of software based on a usage model. *Softw. Pract. Exper.*, 25(1):97–108, 1995.

[Elbaum, Malishevsky, Rothermel 2002] Test case prioritization: A family of empirical studies. *IEEE Transactions on Software Engineering*, 28(2):159–182, 2002.

[Amland 2000] Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study. *Journal of Systems and Software*, 53(3):287–295, 2000.

36

Backup Slides

37

Automated Test Case Generation Using Classification Trees

Peter M. Kruse & Magdalena Luniak
Berner & Mattner Systemtechnik GmbH
Berlin, Germany
{peter.kruse | magdalena.luniak}@berner-mattner.com

Abstract

The basic problem in software testing is choosing a subset from the near infinite number of possible test cases. Testers must select test cases to design, create, and then execute. Due to limited test resources, it is important to select the best possible set of tests. In this paper, we present test case design with the Classification-Tree Editor (CTE XL), the most popular tool for systematic black-box test case design of classification tree-based tests. We show how to integrate weighting factors into classification trees and automatically generate prioritized test suites. In addition to "classical" approaches such as minimal combination and pair-wise, weighted counterparts have been added, along with statistical testing and new generation rules. The upcoming version of CTE XL supports prioritization by occurrence probability, error probability or risk.

Introduction

Software testing is unfortunately one of the activities in which too few resources are invested. Considering a common industry-driven software development, there is often not enough time for testing since the software under test is not finished in time and the release date cannot be delayed. Ideally project teams should re-adjust the planning for their software projects to allow more time to be spent on testing, however the product launch would be delayed too.

In the real world, software testers have to deal with multiple challenges: their resources are limited, deadlines arrive quickly and the system under test is finished too late. Using the software testing period as a buffer for any kind of delay in other phases of software and product development, as well as shortening testing efforts to fulfill the planed product launch date is not uncommon.

Testers have to handle this situation. If there is not enough time to carry out all test cases, a tester can try to finish as many tests as possible. If promising test cases have been determined from previous software projects, a tester would start with these test cases, maybe accompanied by the first *n* test cases from a global test list. There is no guarantee that the selected test cases are more important than any another selection of test cases, however there is no way of identifying important test cases from a given test suite.

In this paper we present an approach for test suite optimization using the classification tree method. We introduce weighting of classification tree elements, allowing us to generate prioritized test suites. These test suites can be optimized by selecting only subsets of test cases. We introduce coverage criteria for identifying the importance of any single test case under given test aspects.

Foundations

Classification tree method

The classification tree method [1] aims for systematic and traceable test case identification for functional testing over all test steps (e.g. component test, system test). It is based on the category partition method [4], which divides the test domain into disjunctive classes representing aspects of importance of the test object. Applying the classification tree method involves two steps, designing the classification tree and defining test cases.

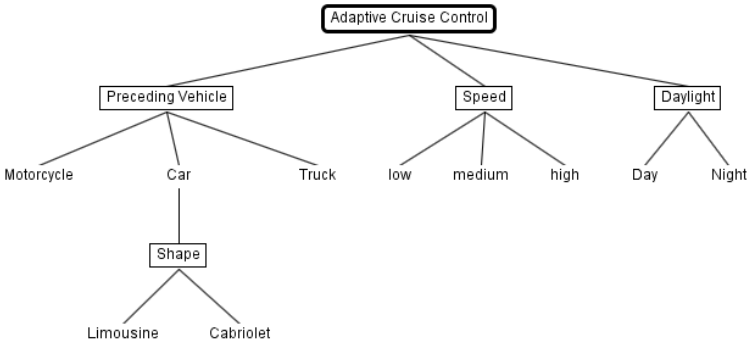


Figure 1 Classification tree for the Adaptive Cruise Control test object

Design of the classification tree

The classification tree is based on the functional specification of the test object. Figure 1 shows an example tree for the *Adaptive Cruise Control* test object. For each aspect of interest (=classifications), the input domain is divided into disjoint subsets (=classes). In our example, classifications are *Proceeding Vehicle*, *Speed* and *Daylight*. Classes for *Speed* are *low*, *medium* and *high*. The class *Car* is further refined into different *Shapes*, which are *Limousine* and *Cabriolet*. Refinements allow test objects to be modeled with any preferred granularity.

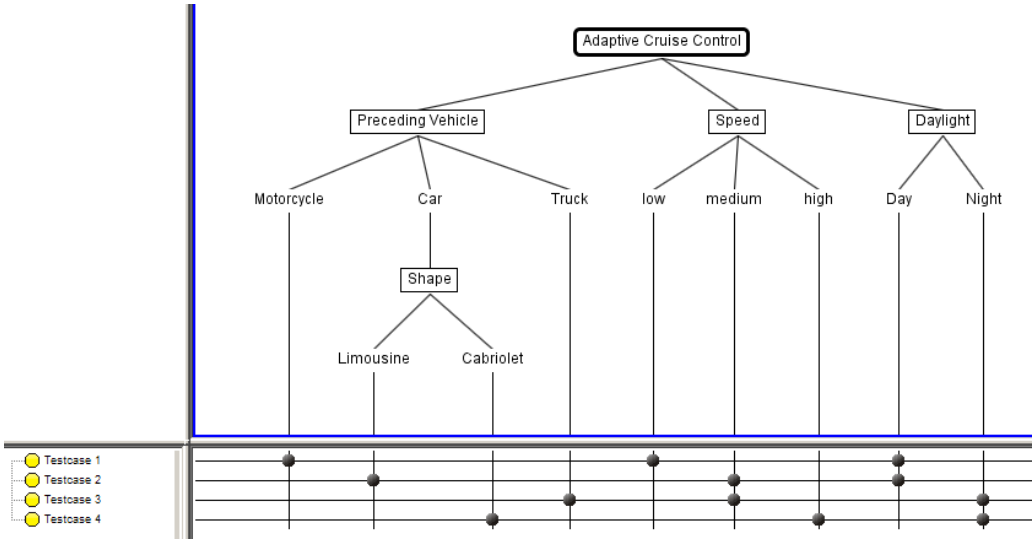


Figure 2 Classification tree with test cases

Definition of test cases

Having composed the classification tree, test cases can be defined by combining classes from different classifications. Figure 2 shows four test cases for the system under test. In *Testcase 1*, the *Adaptive Cruise Control* is tested with a *Motorcycle* as *Preceding Vehicle*, driving *Speed* is *low* and the test is run during the *Day*. Since classifications only contain disjoint values – obviously *Speed* can not be *low* and *high* at the same time – test cases cannot contain several values of one classification.

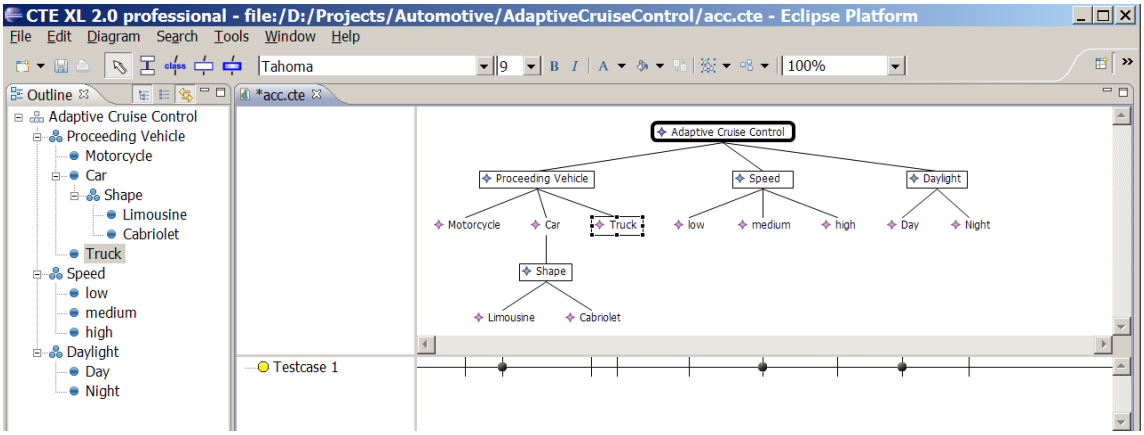


Figure 3 Screenshot of CTE XL

Classification Tree Editor

The Classification Tree Editor (CTE XL) has been introduced together with the classification tree method [2]. A screenshot of CTE XL is provided in Figure 3. Current versions of the CTE XL support automated test case generation, user-defined dependency rules and the integration with requirement and test management tools (e.g. IBM Rational DOORS, HP Quality Center) [3].
Current test case generation offers four different modes:

- Minimal combination creates a test suite that uses every class from each classification at least once in a test case.
- Pairwise combination creates a test suite that uses every class pair from disjunctive classifications at least once in a test case.
- Threewise combination ("*triple-wise*") creates a test suite that uses every triple of classes from disjunctive classifications at least once in a test case.
- Complete combination creates a test suite that uses every possible combination of classes from disjunctive classification in a test case.

Limitations

Earlier versions of CTE XL only support size reduction of test suites by applying different combination rules, e.g. using pairwise combination if complete combination results in too many test cases. Combination rules do not support prioritization of certain test aspects (e.g. costs of an error). The classification tree method does not yet support the weighting of tree elements according to these test aspects. Since they are not taken into account, these aspects cannot be used for comparing the test cases. Additionally, test cases are generated in an arbitrary order, thus, there is no means of selecting an optimized test suite.

Improvements

We have identified the following enhancements:

1. Integration of weights into the classification tree to allow prioritization of certain test aspects
2. Definition of prioritized combination rules using tree weights
3. Introduction of coverage criteria
4. Test suite optimization

Prioritization

Prioritization is used to allow the assignment of values of importance to several classification tree elements. The values of importance are called weights. To cover all kinds of test aspects, these weights can differ. Higher and lower weights should reflect higher and lower importance, respectively. Consequently we are able to compare the elements of the classification tree, to determine their importance under a given test aspect and to prioritize test aspects during test case generation.
We analyzed several existing prioritization techniques. Elbaum *et al* give good overviews of existing approaches [5][6]. The following three models have been selected to provide a basis for prioritization:

- Prioritization based on a usage model [7]: This prioritization tries to reflect usage distribution of all classes in terms of usage scenarios. Classes with high occurrence have higher weights than classes with low occurrence.

- Prioritization based on an error model [5]: This prioritization aims to reflect distribution of error probabilities of all classes. Classes with high probability of revealing an error have higher weights than classes with low probability.
- Prioritization based on a risk model [8]: This prioritization is similar to prioritization based on error model but additionally takes error costs into account. Risk is defined as product of error probability and error costs. Classes with a high risk have higher weights than classes with low risk.

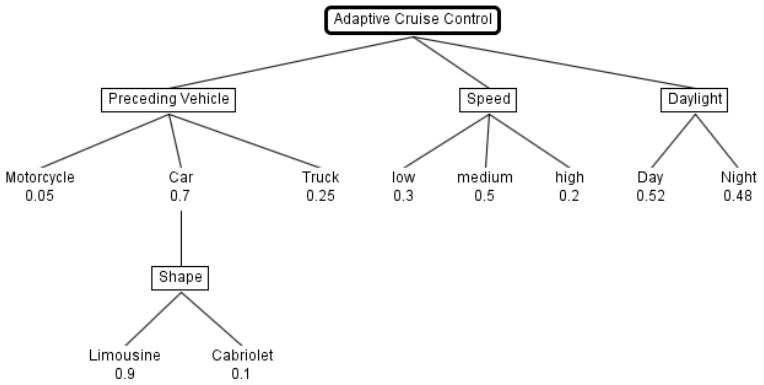


Figure 4 Test object with assigned occurrence values

Example

Our system under test, the *Adaptive Cruise Control*, is shown with assigned occurrence values in Figure 4. All classes have been assigned values of importance. As the figure shows, *medium* is the most probable *Speed*. *Low* and *high* are the subsequent values in descending order of importance.
The weights of all classes composed within one classification sum to 1 in the occurrence model. However, they are independent of each other in both the error model and risk model.
The class *Car* has an occurrence rate of 0.7 of all *Preceding Vehicles*. If the *Preceding Vehicle* is a *Car*, *Limousines* have an occurrence rate of 0.9 and *Cabriolets* have an occurrence rate of 0.1. Refinements are interpreted as conditional probability in the occurrence model. The resulting occurrence probability for a *Limousine* being the *Preceding Vehicle* is 0.63 (= 0.7 * 0.9), for a *Cabriolet* it is 0.07, accordingly.
Since error and risk values are independent of each other, an interpretation as conditional error or risk fails in both the error model and the risk model. All values in these two models are absolute values.

New combination rules and automated test case generation

New combination rules are applied to the weighted classification tree. Existing combination rules are extended and completely new rules are designed. Having implemented combination rules, test suites can be generated automatically.
The existing minimal combination and pairwise combination rules have been extended into prioritized minimal combination and prioritized pairwise combination rules.
We have added class-based statistical combination to enable statistical testing with the classification tree method. All combination rules can are applicable to occurrence, error and risk models.

Prioritized minimal combination

The prioritized minimal combination (PMC) is based on the minimal combination rule. The minimal combination creates a test suite that covers each class in at least one test case, but does not take into account the order of class coverage.
Therefore, we have extended this combination to PMC. Test cases are generated in descending order of importance. Each new test case contains the most important class remaining from all classes in the tree plus as many still uncovered classes as possible. If there are only covered classes left and the test case is not yet completed, the most underrepresented classes are taken. Underrepresentation is calculated by comparing the weight of each class with the ratio of already generated test cases containing this class. This ensures that classes with high weight are used more often during test case composition than classes with low weight.
PMC guarantees coverage of at least the *n* most important classes by the *n* first test cases. We accept that test suites might be slightly larger than generated by plain minimal combinations. As a matter of course, dependency rules are taken into account. For identical trees, the PMC is deterministic and always generates the same test suite.

Prioritized pairwise combination

The plain pairwise combination creates a test suite that covers each class pair in at least one test case. We have extended this combination to a prioritized pairwise combination (PPC).

PPC uses class pairs. The combined weight for class pairs is calculated by multiplying either the occurrence or error probabilities. By contrast, the combined risk is the product of the summed individual costs and the multiplied individual errors. Test cases are generated in descending order of importance.

The most important class pair from all class pairs still uncovered is identified for composing each new test case. Furthermore, we determine all candidate test cases containing this class pair and calculate index values for all candidates. This index value includes the weights and the number of newly covered class pairs. It emphasizes class pairs with a high weight. In this way PPC selects the test case with the highest assigned index value.

PPC guarantees coverage of at least the *n* most important class pairs by the *n* first test cases. The generated test suite may be slightly larger than the result of the plain pairwise combination because of weights and dependencies taken into account. The generation process using PPC is deterministic: the same test suite is generated for identical trees.

Example

The combined occurrence values for class pairs are shown in Table 1. The pair with the highest value is *Limousine & Day*, followed by *Limousine & Night*. We would therefore expect to see these pairs in the first two test cases of a generated test suite. Since classes from the same classification cannot be combined, there are cells without values in the table.

Table 1 Combined class pair values for occurrence model

		Day	**Night**	**low**	**medium**	**high**
		0.52	0.48	0.3	0.5	0.2
Motorcycle	0.05	0.026	0.024	0.015	0.025	0.01
Limousine	0.63	0.3276	0.3024	0.189	0.315	0.126
Cabriolet	0.07	0.0364	0.0336	0.021	0.035	0.014
Truck	0.25	0.13	0.12	0.075	0.125	0.05
low	0.3	0.156	0.144	/	/	/
medium	0.5	0.26	0.24	/	/	/
high	0.2	0.104	0.096	/	/	/

The expected result of a test suite generated with PPC with respect to the occurrence model is given in Table 2. The algorithm creates 13 test cases. The first test case covers the pair *Limousine & medium* as expected, followed by a test case containing *Limousine & Night*.

Table 2 Test cases for PPC occurrence

#	**Preceding Vehicle**	**Speed**	**Daylight**
1	Limousine	medium	Day
2	Limousine	medium	Night
3	Limousine	low	Day
4	Truck	low	Night
5	Truck	high	Day
6	Limousine	high	Night
7	Truck	medium	Night
8	Cabriolet	medium	Day
9	Cabriolet	low	Night
10	Motorcycle	medium	Day
11	Motorcycle	low	Night
12	Cabriolet	high	Night
13	Motorcycle	high	Night

Class-based statistical combination

The class-based statistical combination (CSC) rule is a completely new combination rule for statistical testing in the classification tree method. CSC generates test suites reflecting the distribution of classes defined by a priority

model. The tester obtains a test suite with a given size that reflects the test aspect well. Test cases are generated using a random process. For each classification in the tree, one class is randomly selected according to its distribution within the classification.
Test suites created using CSC may contain redundant test cases. With CSC, the classification tree method allows for testing of non-deterministic systems. The repeated execution of test cases increases the chance of revealing errors in the system.

Coverage criteria and optimization of test suites

To assess the test progress, coverage criteria are commonly used in the testing process. Therefore we introduce coverage criteria for prioritized combination rules.
The each used coverage (EUC) for PMC measures the coverage of classes in a test suite. It only takes those classes into account that can be covered by test cases and are not excluded by dependencies.

$$EUC_{PMC} = \frac{number\ of\ covered\ classes}{number\ of\ coverable\ classes}$$

The weight coverage (WC) measures the coverage of weights in a test suite. Again, we only take into account classes that are coverable.

$$WC_{PMC} = \frac{sum\ of\ weights\ of\ covered\ classes}{sum\ of\ weights\ of\ coverable\ classes}$$

For the PPC, we defined the pair coverage metric, measuring the coverage of new pairs in the test suite.

$$EUC_{PPC} = \frac{number\ of\ covered\ class\ pairs}{number\ of\ coverable\ class\ pairs}$$

The weight coverage can be calculated in the same way.

$$WC_{PPC} = \frac{sum\ of\ weights\ of\ covered\ class\ pairs}{sum\ of\ weights\ of\ coverable\ class\ pairs}$$

Both metrics are relative since they ignore class pairs that are not coverable due to user-defined dependencies.
We do not define coverage criteria for the class-based statistical combination since test generation is based on a random process, and we cannot assure any coverage. Nevertheless, we offer a chi-square test to measure the quality of the generated test suite. It assesses the significance level at which the distribution of classes in the test suite does not differ from the class distribution assumed by the priority model.

Table 3 Test cases with coverage values

#	Preceding Vehicle	Speed	Daylight	EUC	WC
1	Limousine	medium	Day	0.12	0.3
2	Limousine	medium	Night	0.19	0.48
3	Limousine	low	Day	0.27	0.6
4	Truck	low	Night	0.38	0.71
5	Truck	high	Day	0.5	0.8
6	Limousine	high	Night	0.58	0.88
7	Truck	medium	Night	0.62	0.92
8	Cabriolet	medium	Day	0.69	0.94
9	Cabriolet	low	Night	0.77	0.96
10	Motorcycle	medium	Day	0.85	0.98
11	Motorcycle	low	Night	0.92	0.99
12	Cabriolet	high	Night	0.96	0.99
13	Motorcycle	high	Night	1	1

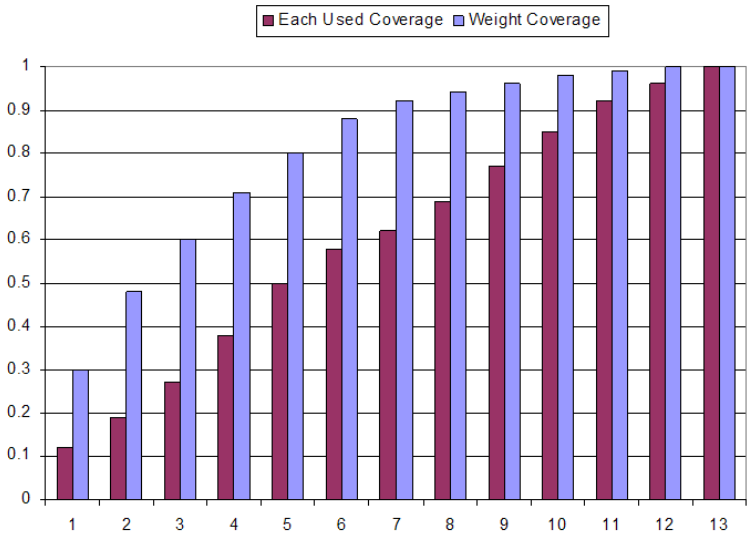


Figure 5 Each used coverage vs. weight coverage

Example

For prioritized pairwise combination, Table 3 and Figure 5 show that the weight coverage grows faster than the each used coverage for the first six test cases. In the last five test cases, the weight coverage increases slower since the class pairs with high priority have already been covered at the beginning of the generation process. The whole test suite guarantees both 100% of the each used and the weight coverage.

The test suite optimization allows us to achieve a defined coverage, while minimizing the size of the test suite. Only using the first test case, we already obtain 30% of the weight coverage. The first three test cases reach 60% weight coverage.

Implementation

After identifying improvements to the classification tree method, we have integrated them into the new version of CTE XL.

Prioritization

To include weights for classes, the tester has to assign values to the classification tree. Models can be used independently of each other; it is possible to only use occurrence probabilities by assigning only occurrence values or to use all three models consecutively.
To enable the usage of a priority model, the tester assigns values to all classes as shown in Figure 6. The class *Truck* has an occurrence probability of 25%. In the screenshot, error and risk values have been entered as well: the error probability is 5% and costs of an error are 40000, resulting into a risk of 2000.

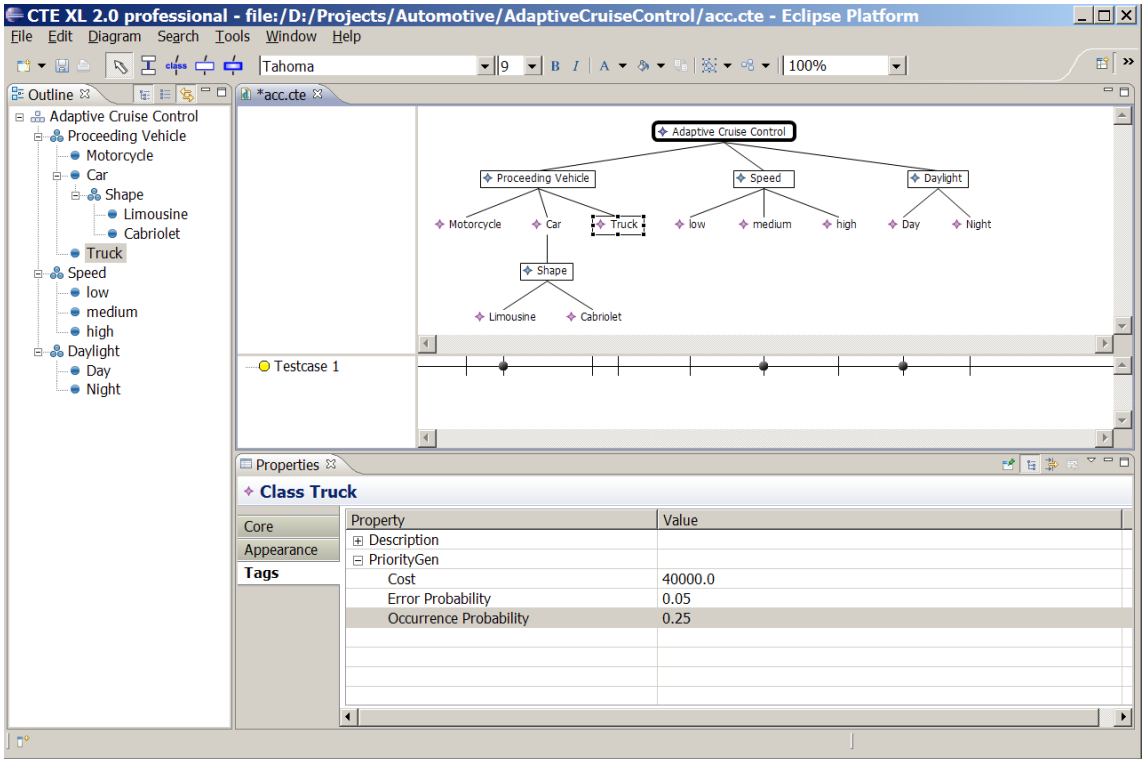


Figure 6 Prioritization (Screen Shot)

New combination rules and automated test case generation

With all values assigned, the user can start prioritized test case generation. The test case generation dialog provides all available combination rules (see Figure 7). Priority models are available only if respective values have been assigned to all classes.

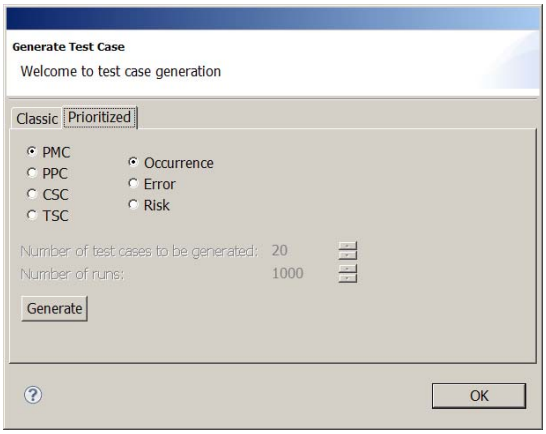


Figure 7 Combination rules (Screen Shot)

Prioritized combinations do not need any further parameters. For statistical combinations (e.g. CSC), the tester can specify the number of test cases to generate and the number of tries.
The complete result for PPC occurrence can be found in Figure 8. The order of test cases conforms to Table 2.

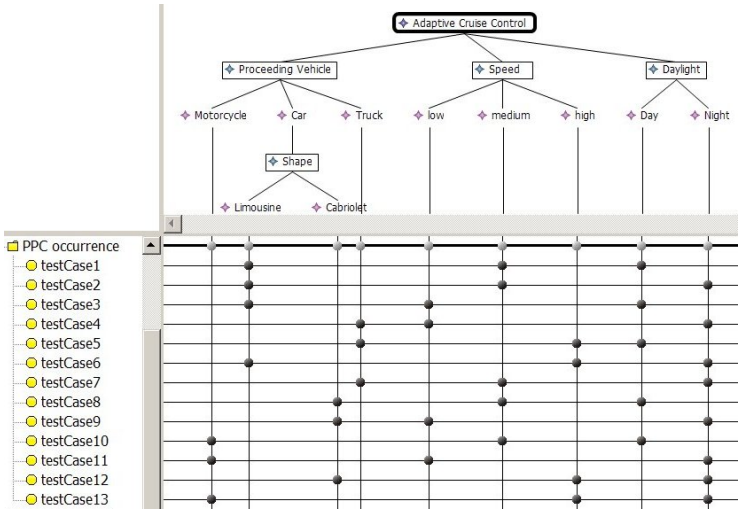


Figure 8 Test suite generated for PPC using occurrence values

Coverage criteria and optimization of test suites

The support for coverage criteria has been added to the CTE XL, too. Using the optimization dialog (Figure 9), the tester can optimize the generated test suite by selecting the preferred coverage criteria and adjusting the coverage level. A preview of the resulting test suite is shown, together with the number and ratio of test cases.

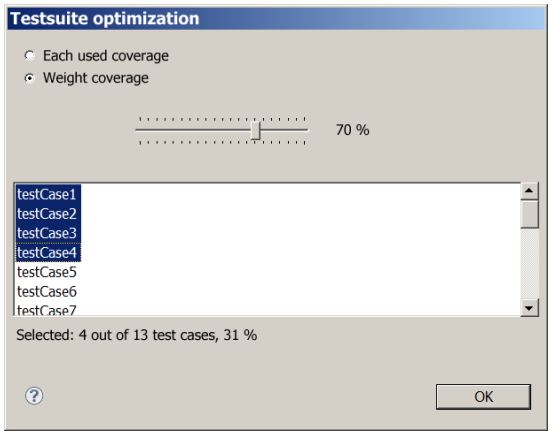


Figure 9 Testsuite optimization in CTE XL

Conclusion and Future Work

We developed and implemented the integration of weights into the classification tree to allow prioritization with respect to test aspects. Prioritized combination rules using tree weights have been defined and implemented. Thanks to the introduction of coverage criteria, test suite optimization was enabled.
As a result, testers are now able to prioritize and weight tree elements according to test aspects (e.g. costs of an error). Test aspects are used for comparing the test cases; test cases are generated in a defined order. Testers can now select optimal test suites meeting their individual requirements.
All these enhancements are valuable extensions to the classification tree method and are available in the newest versions of CTE XL.

Future work will concentrate on further integration with testing tools. Moreover, we will extend the remaining plain combinations (threewise and complete combination) in such a way that they take into account priority values and develop further combinations for statistical testing.

References

[1] Grochtmann, M. and Grimm, K. “Classification Trees for Partition Testing,” Software Testing, Verification & Reliability, Volume 3, No 2, 1993, pp. 63-82.

[2] Grochtmann, M., Grimm, K., and Wegener, J. “Tool-Supported Test Case Design for Black-Box, Testing by Means of the Classification-Tree Editor,” Proceedings of EuroSTAR’93, London, Great Britain, 1993.

[3] Lehmann, E. and Wegener, J. “Test Case Design by Means of the CTE XL,” Proceedings of the 8th European International Conference on Software Testing, Analysis & Review (EuroSTAR 2000), Kopenhagen, Denmark. 2000.

[4] Ostrand, T. J. and Balcer, M. J. “The category-partition method for specifying and generating functional tests,” Communications of the ACM, 31(6):676–686, 1988.

[5] Elbaum, S. and Malishevsky, A.G. and Rothermel G., “Test Case Prioritization: A Family of Empirical Studies,” IEEE Transactions on Software Engineering, 28(2):159–182, 2002.

[6] Elbaum, S. and Rothermel, G. and Kanduri, S. and Malishevsky, A.G., “Selecting a Cost-Effective Test Case Prioritization Technique,” Software Quality Journal, 12:185–210, 2004.

[7] Walton, G.W., and Poore, J. H. and Trammell, C.J., “Statistical testing of software based on a usage model,” Softw. Pract. Exper., 25(1):97–108, 1995.

[8] Amland, S., “Risk-based testing: Risk analysis fundamentals and metrics for software testing including a financial application case study,” Journal of Systems and Software, 53(3):287–295, 2000.